Une Approche Formelle pour la Génération d'Analyseurs de Langages Naturels

Manuel Vilares Ferro Alberto Valderruten Vidal
Jorge Graña Gil Miguel A. Alonso Pardo

Résumé

Un processus d'analyse syntaxique et d'étiquetage efficace est déterminante dans l'élaboration de structures d'analyse de langages naturels. Ce papier introduit un environnement de dévéloppement permettant l'implémentation du support formel des langages naturels à partir de deux points de vue, analyse syntaxique et étiquetage. Le problème de l'analyse syntaxique repose sur l'analyse de grammaires algébriques sans restrictions, et celui de l'étiquetage sur des automates finis non déterministes.

L'analyseur syntaxique prends en entrée un texte arbitraire, et suit la structure désignée par une grammaire algébrique. Le partage syntaxique est optimisé de façon à favoriser l'élimination des ambigüités pendant le processus sémantique. Les automates à états finis sont utilisés comme formalisme opérationnel pour étiqueter les corpora de façon efficace, spécialement pour les langages dont l'analyse morphologique a une relevance accrue. Les deux activités, analyse syntaxique et étiquetage, sont intégrées dans un même outil, Galena[1]*, fournissant l'incrémentalité comme fonctionnalité favorisant la réutilisabilité des composantes d'un point de vue génie logiciel.*

Mots clés: *Analyse Syntaxique, Analyse Morphologique, Automates à États Finis, Automates à Piles, Forêt Partagée, Étiquetage.*

1 Introduction

L'utilisation de grammaires algébriques pour la description de langages naturels a pris de l'intérêt du point de vue de la lingüistique théorique pendant ces dernières années. Ces grammaires sont adaptées à l'implémentation, et montrent qu'avec leur utilisation les systèmes d'analyse de langages naturels peuvent incorporer des composantes à la fois efficaces du point de vue du traitement et élégantes du point de vue lingüistique, à l'opposé des approches heuristiques des systèmes conventionnels. Nous ne suggérons pas qu'il existe une grammaire algébrique pour décrire chaque langage naturel; il est probablement plus indiqué de voir la grammaire comme une structure de contrôle guidant l'analyse du texte en entrée, l'analyse sémantique postérieure étant motivée par un modèle lingüistique plus sophistiqué.

La section 2 décrit le fonctionnement du module d'étiquetage du système. Le générateur d'analyseurs syntaxiques est présenté dans la section 3. Les analyseurs syntaxiques générés peuvent être utilisés suivant deux modes de fonctionnement: standard et incrémental, décrits respectivement dans les sections 4 et 5. Après une description générale de l'interface utilisateur dans la section 6, nous déroulons dans la section 7 un exemple concret afin de montrer le fonctionnement du système. Les statistiques et les performances du système sont analysées dans la section 8. Finalement, les conclusions et les perspectives font l'objet de la section 9.

M. Vilares, J. Graña et A. Valderruten: Laboratorio de Fundamentos de la Computación e Inteligencia Artificial, Departamento de Computación, Universidad de A Coruña, Campus de Elviña, 15071 A Coruña, España. E. mail: vilares, grana, valderruten@dc.fi.udc.es.

M. A. Alonso: Centro de Investigaciones Lingüísticas y Literarias Ramón Piñeiro, Carretera Santiago-Noya, Km. 3, A Barcia, 15896 Santiago de Compostela, España. E. mail: alonso@dc.fi.udc.es.

Ce travail a été partiellement financé par le projet **E**ureka **S**oftware **F**actory, et par le Gouvernement Autonome de Galice, projet XUGA10501A93.

[1] Pour **G***enerador de* **A***nalizadores para* **Le***nguages* **Na***turales.*

2 L'analyse lexicale

L'analyse lexicale constitue la première phase dans le processus informatique des langages naturels, comprenant l'étiquetage des mots ainsi que la détection et l'élimination des possibles ambigüités. A ce propos, notre travail prétend donner un traitement essentiellement pratique au problème, fixant notre attention sur les langages ayant une composante morphologique importante[2].

Pour répondre à ces besoins, nous avons pris comme point de départ l'expérience accumulée pendant des années dans le traitement du même problème au niveau des langages formels. Cette approche implique l'utilisation d'un même concept, celui des automates finis, à la fois comme formalisme opérationnel et comme formalisme de description de la structure morphologique du langage.

Pour chaque mot du texte en entrée, notre analyseur fournit, dans son état actuel, une étiquette avec les champs: Catégorie, Type, Genre, Nombre, Mode Verbal, Temps Verbal, Personne, Détermination, Cas, Comparaison et Forme Canonique. La catégorie est le champ principal, et elle détermine l'existence de valeurs significatives dans les autres champs. Les différentes catégories reconnues sont: Adjectif, Adverbe, Conjonction, Déterminant, Nom, Phrase Faite, Ponctuation, Préposition, Pronom, Verbe et Verbe Être. D'autre part, le champ Type est une spécialisation de la Catégorie, ainsi par exemple, nous povons différencier les Noms Communs des Propres; les Conjonctions Coordonées des Subordonnées; les Adverbes Nucléaires des Modifieurs ou des Relatifs; ou les Déterminants Articles des Démonstratifs, Possessifs, Ordinaux ou encore des Cardinaux, etc. Le système fournit à ce niveau une interface graphique spécialisée permettant l'utilisateur de maintenir facilement la base lexicale.

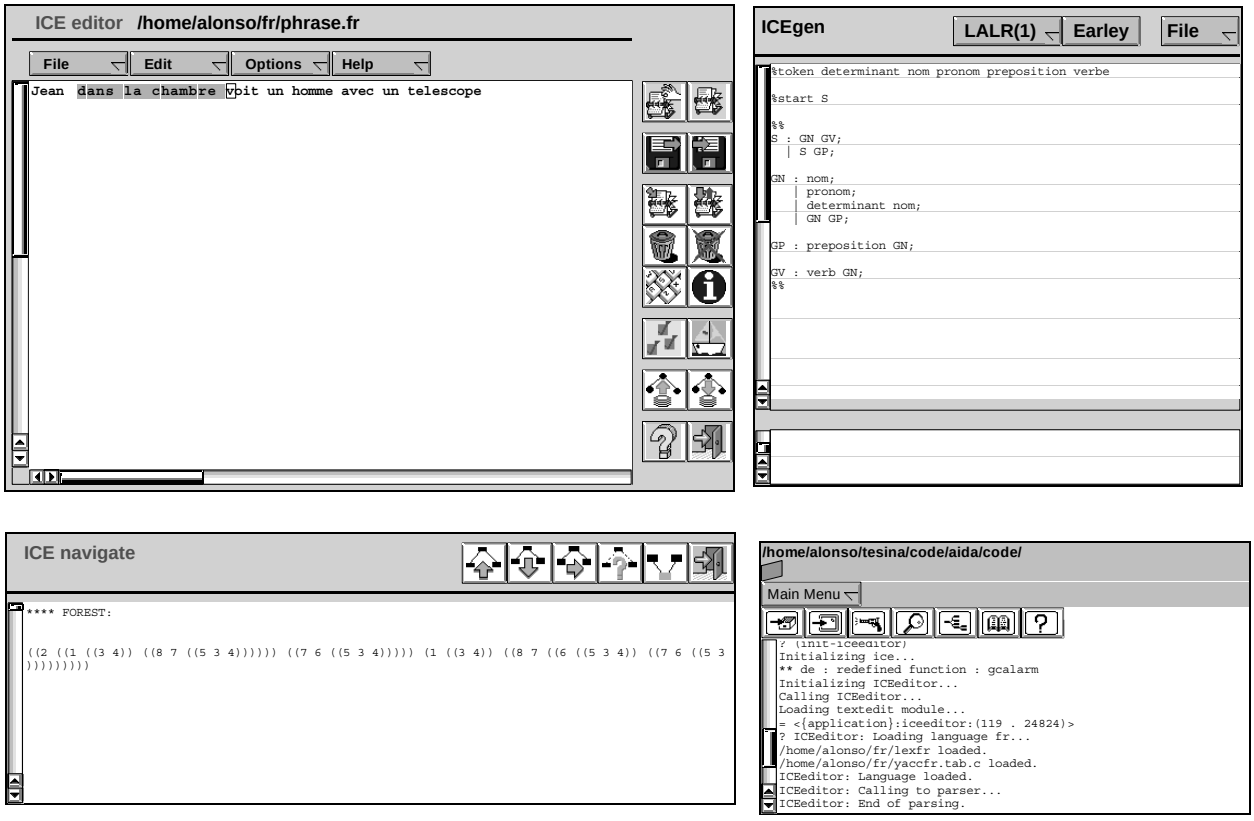

Figure 1: Environnement d'analyse de textes

[2] Tel est le cas des langages d'origine latine en général, du Français en particulier.

3 Le générateur d'analyseurs syntaxiques

Le but de cette section est de présenter les principes utilisés pour la génération d'analyseurs syntaxiques pour les langages naturels, basée sur l'approche de ICE [1], un environnement de dévéloppement de langages algébriques arbitraires.

Notre but est de proposer un environnement de génération et validation de langages. Dans ce contexte les grammaires évoluent, car elles ne sont pas encore fixées, ce qui demande une vitesse accrue de la part du générateur. Il est également important d'assurer la portabilité de l'outil, ce qui est assuré au niveau de l'implémentation par le choix du langage C et au niveau de l'utilisation par l'incorporation d'un formalisme standard de description de grammaires, celui de BISON [2], un des plus performants générateurs d'analyseurs de langages de programmation sur UNIX.

De façon à favoriser la conception de langages, il faut assurer une certaine flexibilité dans le choix du type d'analyseur à générer. Nous proposons deux options qui se correspondent à des approches fondamentales dans le traitement des langages naturels: les analyseurs dirigés par des grammaires et les analyseurs supportés par des automates.

Finalement, il faut permettre une validation aisée du langage généré, en supportant des étapes successives de correction et d'analyse de textes d'entrée. À ce propos, notre outil inclut une fonctionnalité incrémentale générique applicable au processus d'analyse syntaxique, indépendante du type d'analyseur généré par le système.

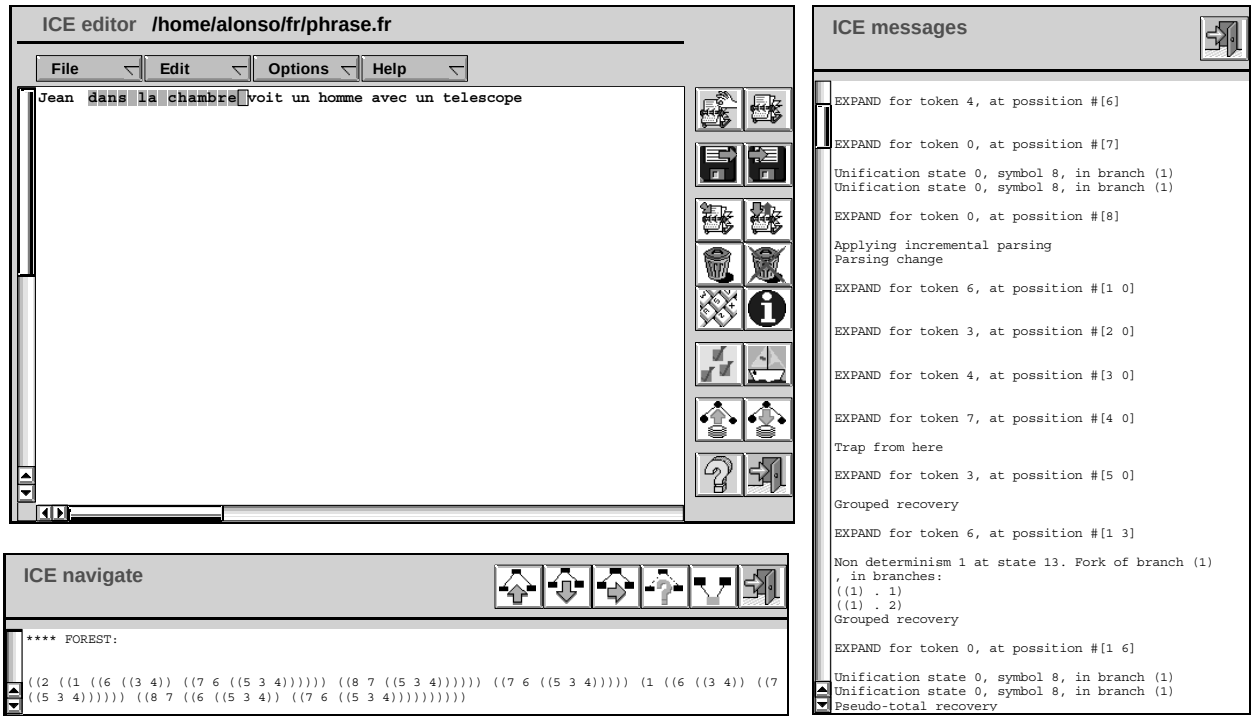

Figure 2: Environnement d'analyse de textes

4 L'analyse syntaxique standard

Avec GALENA, l'utilisateur peut choisir parmi deux schémas d'analyse syntaxique, la méthode d'Earley [3] et un schéma LALR(1) étendu. Afin d'illustrer la discussion qui suit, considérons la mini-grammaire $\mathcal{G}$ suivante:

(0)	$\Phi \rightarrow$ S $\dashv$	(1)	S $\rightarrow$ GN GV	(2)	S $\rightarrow$ S GP
(3)	GN $\rightarrow$ nom	(4)	GN $\rightarrow$ pronom	(5)	GN $\rightarrow$ déterminant nom
(6)	GN $\rightarrow$ GN GP	(7)	GP $\rightarrow$ préposition GN	(8)	GV $\rightarrow$ verbe GN

4.1 Le modèle descriptif

Une différence apparamment majeure de notre approche face à la plupart des analyseurs syntaxiques vient de la structure de sortie choisie pour représenter la forêt partagée. En effet, nous utilisons une grammaire algébrique comme structure de sortie.

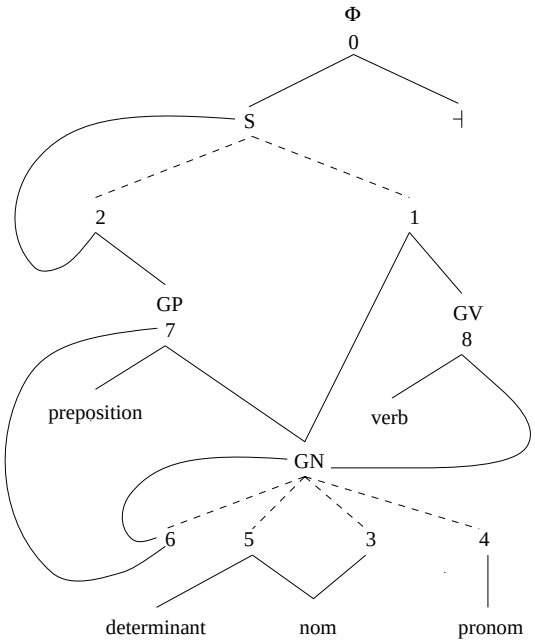

Figure 3: Graphe ET-OU de la mini-grammaire $\mathcal{G}$

Cependant, cette différence est seulement apparente. Les grammaires algébriques peuvent être représentées par des graphes ET-OU, proches des représentations syntaxiques classiques [11]. Plus précisément, n'importe quel graphe ET-OU tel que toutes les étiquettes des noeuds sont différentes, peut être traduit sous forme de grammaire algébrique où chaque étiquette d'un noeud ET représente une règle, et chaque étiquette d'un noeud OU représente une catégorie syntaxique. Les étiquettes des feuilles sont quant à elles les catégories lexicales.

Le formalisme utilisé permet de partager la forêt résultante de l'analyse syntaxique de façon optimale, ce qui peut être prouvé formellement. Comme exemple, nous montrons une possible représentation de notre mini-grammaire $\mathcal{G}$ dans la Figure 3.

4.2 Le modèle opérationnel

L'algorithme d'Earley est une méthode classique d'analyse dirigée par une grammaire algébrique, qui utilise les techniques de programmation dynamique. Celles-ci permettent à l'algorithme de ne pas atteindre la complexité maximale, de même pour l'espace comme pour le temps, dans la plupart des cas pratiques, ce qui représente une avantage fondamentale face aux algorithmes de type CKY [4, 5, 6] ayant seulement un intérêt asymptotique [7, 8]. Essentiellement, Earley associe à chaque symbole du texte d'entrée un ensemble de "règles pointées" contenant la position dans la production et l'endroit où nous avons commencé à la reconnaître. Ce type d'algorithme s'adapte facilement aux changements de la grammaire puisqu'il n'a pas une phase préliminaire de traitement de celle-ci. Cependant, pour chaque étape d'analyse toute l'information doit être trouvée à partir de la grammaire; malgré ce manque d'efficacité, la méthode d'Earley a été étendue pour la reconnaissance de la parole [9] et pour la génération d'interprètes logiques [10]. Simple et puissant, cet algorithme est la référence de toute une école dans le domaine du traitement de langages algébriques [11, 12, 13] et plus généralement naturels [14, 15], ce qui justifie son inclusion dans notre système.

La construction dynamique considérée par Earley peut être naturellement étendue aux modèles de traducteurs à piles [11], afin d'éviter les problèmes dérivés des contraintes grammaticales de la méthode originale. L'idée consiste à séparer la stratégie d'exécution de l'implémentation du

traducteur. Nous pouvons ainsi obtenir une famille d'analyseurs syntaxiques parallèles pour chaque famille d'analyseurs syntaxiques déterministes, en simulant tous les calculs avec une complexité du même ordre que celle d'Earley. Notre expérience [16] a montré que les modèles de traducteurs à piles proches aux méthodes ascendantes les plus simples telles que LALR(1) sont les plus appropriés pour implémenter ce type d'analyseur, ce qui est en concordance avec les résultats obtenus par des travaux précédents [17, 18]. D'autres modèles plus performants dans le cas déterministe tels que LR(k), $k \geq 1$, ou encore LALR(k), $k > 1$, ne conservent pas leur efficacité lorsqu'ils sont étendus au non-déterminisme. En effet, ce gain d'efficacité a son origine dans des mécanismes qui réduisent considérablement le domaine déterministe en multipliant les schémas d'analyse, ce qui empêche le partage de calculs identiques[3]. Ceci a comme effet une inefficacité accrue dans le traitement de phénomènes de déterminisme local[4], d'où l'inclusion de la méthode LALR(1) étendue dans notre système.

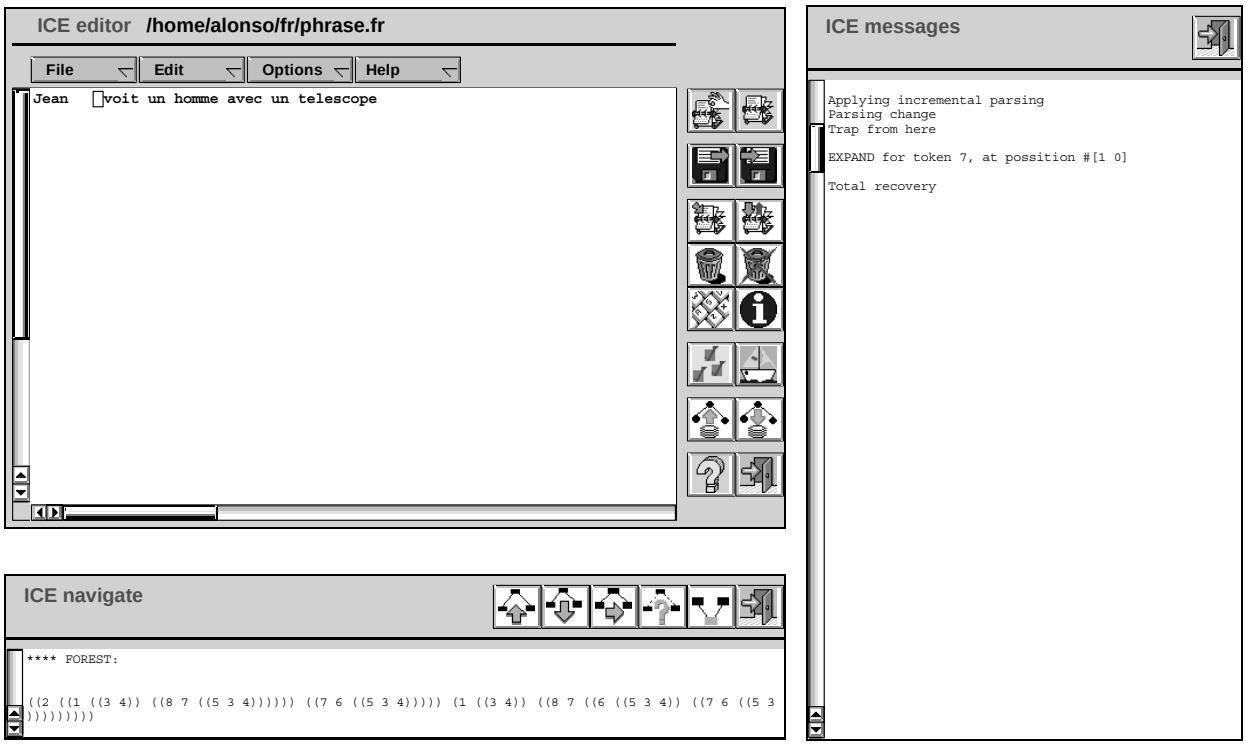

Figure 4: Environnement d'analyse de textes

5 L'analyse syntaxique incrémentale

L'outil inclut une fonctionnalité incrémentale pour l'analyse syntaxique [19] qui assure la reconstruction des sous-arbres avec le même niveau de partage fourni par le mode non incrémental. Ce choix se justifie par le besoin d'un traitement efficace lorsque des corrections dans le texte d'entrée ont été réalisées.

Puisque nous nous intéressons au traitement de mises à jour arbitraires dans le texte, il faut analyser de quelle façon elles peuvent affecter la forêt partagée. Ce qui nous amène à considérer les quatre cas de récupération incrémentale qui sont schématisés dans la Figure 5:

- *Récupération Totale*, lorsque l'analyse devient indépendante des modifications dans ce qui reste à analyser.

[3] On prend en compte des contextes irrélevants d'analyse syntaxique, typiquement en relation à l'utilisation de *look-aheads* ou de techniques prédictives pour la génération de l'automate.

[4] Les ambigüités ont souvent un comportement local puisque l'utilisateur écrit la plupart du temps des messages non ambigus, et une analyse déterministe de cette partie du texte est donc possible.

- *Récupération Partielle.* Nous y distingons trois cas. D'abord, lorsque la récupération est possible pour toutes les branches sur un intervalle propre du texte pas encore analysé, nous parlons de *récupération groupée.* Deuxièmement, si la récupération est possible dans ce qui reste à analyser, mais seulement pour quelques branches de la forêt, nous parlons de *récupération isolée.* Finalement, la récupération peut être possible pour quelques branches, dans un intervalle propre du texte pas encore analysé; il s'agit de la *récupération sélective.*

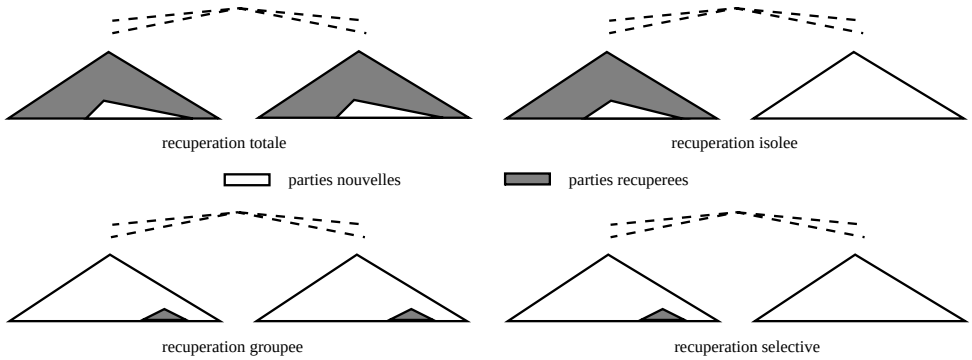

Figure 5: Types de récupération incrémentale

Même si tous les cas de récupération incrémentale ont été étudiés, l'expérience montre que ce traitement n'est intéressant que si toutes les branches de la forêt peuvent être récupérées dans un intervalle donné du texte d'entrée, cas le plus courant lorsque la grammaire possède un nombre raisonnable d'ambigüités. En conséquence, nous considérons seulement la récupération incrémentale dans les cas totale et groupée. Le critère essentiel justifiant ce choix est l'efficacité. En effet, récupérer un ensemble propre de branches dans un intervalle de texte est équivalent à récupérer des arbres isolés de la forêt dans un noeud ambigu. Afin de garantir un partage optimal de la forêt résultante de l'analyse incrémentale, il serait nécessaire de mettre en place un algorithme plus complexe pour la reconstruction de la forêt. Ceci implique un coût temporel trop élevé. Dans la pratique, les ambigüités sont bien localisées et ce type de phénomène est limité.

Finalement, un dernier aspect important que nous avons abordé est la possible influence de l'utilisation de la fonctionnalité incrémentale sur les performances du processus d'analyse en mode standard. À ce propos, les tests ont prouvé que les performances ne sont pratiquement pas affectées par les vérifications additionelles requises par le mode incrémental.

6 L'interface utilisateur

Le système Galena possède une interface utilisateur multi-fenêtre à menus déroulants [20] implémentée sur X11. L'interface utilisateur associée au générateur d'analyseurs unifie l'apparence des algorithmes de génération disponibles dans le système. L'outil propose un éditeur dédié, ICEgen, pour l'écriture de la grammaire avec le même formalisme de Bison [2]. Ainsi à tout moment, à partir de cette description l'utilisateur peut demander la compilation de la grammaire suivant un schéma choisi. Les fenêtres d'édition peuvent également être personnalisées; l'utilisateur dispose ainsi d'un moyen pour ajouter, par exemple, un nouvel éditeur plus spécialisé.

L'interface correspondante à l'environnement d'édition de textes, ICEeditor, offre la possibilité de choisir parmi les différents algorithmes d'analyse disponibles sur le système et de charger un langage généré précédemment. L'utilisateur peut éditer un texte et l'analyser en invoquant l'analyseur correspondant sous l'un des modes disponibles: standard ou incrémental. Le système garantit, dans tous les cas, un niveau optimal de partage de la forêt syntaxique résultante. De plus, un ensemble d'options déterminent le type d'information reportée par le système: ambigüités détectées, statistiques sur la quantité de travail produite, etc. Les fonctionnalités de trace incluent également des informations sur le processus incrémental en permettant ainsi des tests comparatifs entre différents algorithmes d'analyse. De la même façon, les erreurs sont reportées. D'autre part, l'interface utilisateur permet de récupérer et de parcourir facilement la forêt.

Afin de disposer d'un outil plus convivial, nous avons prévu la possibilité de sélectionner la

langue des commandes parmi l'Espagnol, le Galicien[5], le Français et l'Anglais. Également, des fonctionnalités d'aide sont disponibles à tout moment afin de répondre aux possibles questions sur l'utilisation du système. Finalement, les ressources graphiques telles les polices de caractères ou les couleurs sont exploitées au niveau sémantique pour l'édition[6].

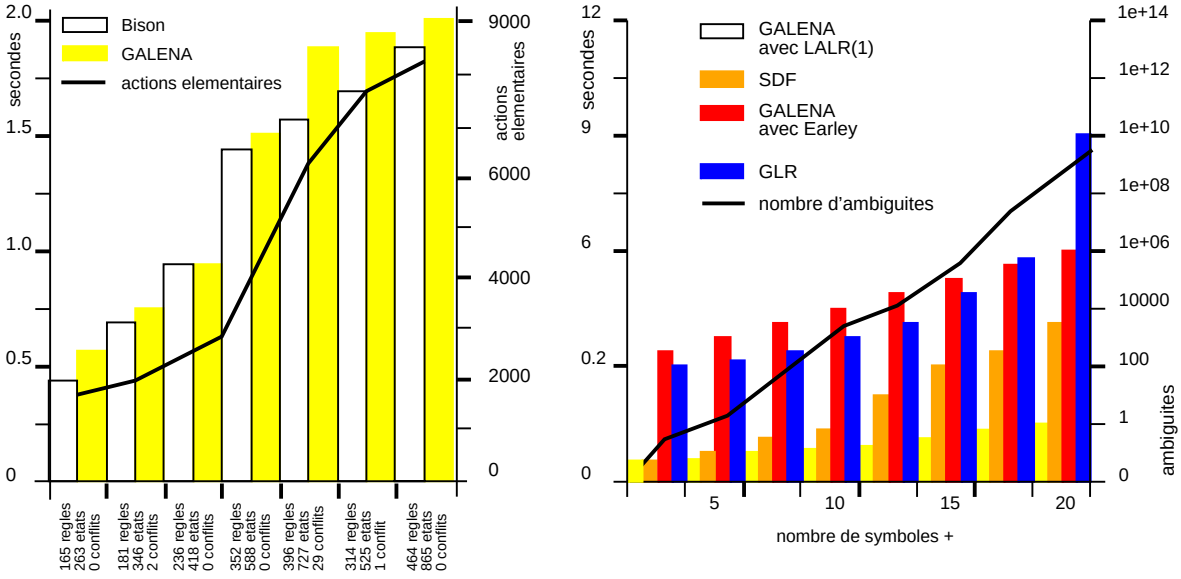

Figure 6: Temps de génération d'analyseurs et temps d'analyse

7 Le fonctionnement du système

Afin d'expliquer le comportement de notre environnement pour une session de travail, nous allons construire un analyseur pour le langage défini par la grammaire $\mathcal{G}$ de notre exemple. Nous faisons ici référence aux Figures 1, 2 et 4, contenant les fenêtres de l'interface graphique à différents moments du processus.

Dans la première étape, nous construisons un fichier décrivant la grammaire en utilisant l'outil ICEgen. La fenêtre principale de cet outil, présentée en haut à droite de la Figure 1, contient l'éditeur pour introduire les règles de la grammaire. Tous les éditeurs utilisés dans le système sont inspirés par EMACS [21], éditeur standard sur UNIX.

Pour compiler la grammaire, nous disposons d'un bouton pour chacun des algorithmes de génération d'analyseurs disponibles. Les messages d'erreur de la compilation sont montrés dans la sous-fenêtre en bas de ICEgen. En cliquant sur un message, l'éditeur se positionne sur la ligne qui a provoqué l'erreur. S'il n'y a pas d'erreur, l'analyseur est généré. À ce moment, la phase de validation du langage peut commencer.

L'outil ICEeditor apparaît en haut à gauche des Figures 1, 2 et 4. Des boutons sont prévus pour les opérations élémentaires telles l'insertion, l'élimination ou le remplacement de texte, en plus de l'annulation des dernières éditions. À chaque fois qu'un texte est analysé, une fenêtre contenant les messages produits par l'analyse est activée, appelée ICEmessages dans les Figures 2 et 4. L'utilisateur peut choisir le niveau d'information fourni par cette fenêtre, allant du mode "muet" à celui où toute action élémentaire[7] de l'analyse est montrée. Il peut également naviguer dans la forêt partagée, en utilisant l'outil ICEnavigate apparaissant en bas à gauche des Figures 1, 2 et 4. Dans chaque mouvement au niveau de la forêt, des informations additionelles sont fournies au sujet du nombre de fils et d'ambigüités possibles du noeud visité. Si nécessaire, les structures générées pendant ces analyses peuvent être sauvegardées.

[5] Langue officielle, avec l'Espagnol, dans la Communauté Autonome de Galice.

[6] Ainsi une couleur donnée peut être identifiée avec un type d'opération d'édition particulière, par exemple l'insertion.

[7] Même, par exemple, les actions de plus bas niveau des automates LALR(1): empiler et dépiler.

En suivant avec notre exemple, nous avons édité à l'aide de ICEeditor un texte simple afin d'illustrer le processus précédemment décrit, en commençant par l'écran de la Figure 1. Si nous ne spécifions pas de langage, le système prend par défaut celui qui est spécifié par l'extension du fichier contenant le texte d'entrée[8], fr pour les textes écrits avec notre grammaire d'exemple. Sur ce même écran, dans la fenêtre ICEgen, nous pouvons voir l'expression de la grammaire qui a généré le langage.

La fenêtre ICEmessages de la Figure 2 montre une trace partielle du processus d'analyse du texte de la fenêtre ICEeditor, qui présente l'insertion du groupe nominal *"dans la chambre"* par rapport à celui du premier écran. Nous pouvons y observer les branches correspondantes aux deux interprétations possibles de l'analyseur, et aussi comment elles sont unifiées plus tard. De la même façon, nous pouvons voir que dans ce cas la récupération incrémentale partielle est possible une fois la modification analysée. Nous pouvons ensuite demander plus de détail à propos des catégories lexicales du texte[9] à l'aide du bouton prévu à cet effet.

Maintenant, nous allons effacer le groupe nominal précédemment introduit afin de montrer le fonctionnement de la récupération incrémentale totale. La Figure 4 montre le déroulement de ce processus.

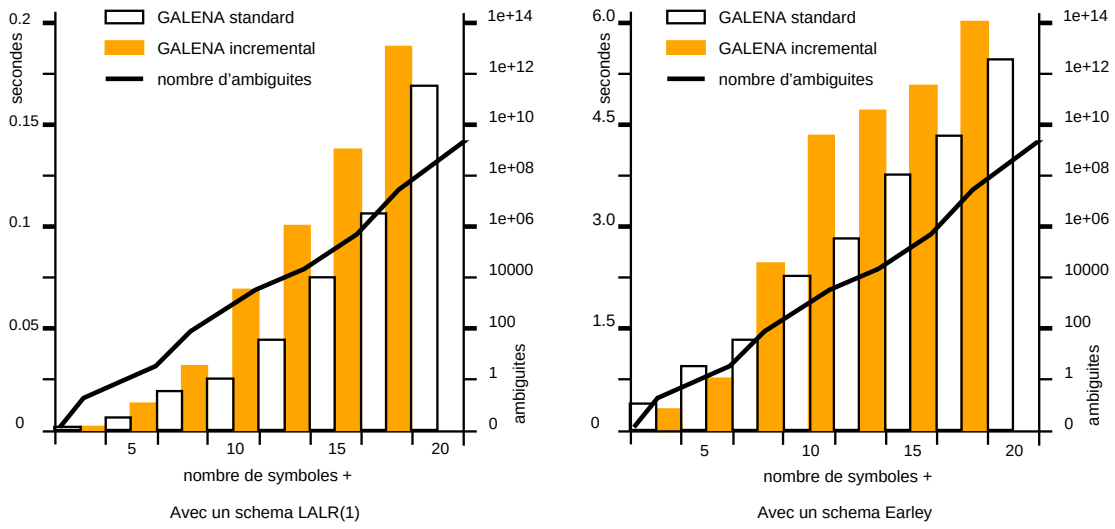

Figure 7: Résultats de l'analyse incrémentale

8 Résultats empiriques

Premièrement, nous nous sommes intéressés aux performances du module d'analyse lexicale. Nous pouvons donner à titre indicatif un temps d'étiquetage mesuré sur un texte en Espagnol littéraire suffisamment représentatif. Avec un extrait de 50.000 mots environ, nous obtenons autour de 20 secondes tout en observant un comportement linéaire selon la taille de l'extrait. Toutes les mesures présentées dans cette section ont été réalisées sur une *Sun SPARCstation 10* dédiée.

D'autre part, nous avons comparé GALENA avec les environnements d'analyse syntaxique à notre avis les plus efficaces, d'après deux points de vue différents: la génération d'analyseurs et le processus d'analyse. Nous montrons également l'efficacité de l'analyse incrémentale par rapport au mode standard.

En ce qui concerne la génération d'analyseurs, nous avons comparé l'outil de génération de langages de programmation BISON [2] avec le générateur LALR(1) de GALENA[10], ce qui implique que nous nous mettons en desavantage au niveau de la complexité des tests puisque le processus

[8] Dans le cas où le texte vient d'être introduit via l'éditeur, il faut donner explicitement le nom du langage.

[9] Par exemple, le contenu des étiquettes correspondantes au mot en question, ainsi que sa position par rapport au texte.

[10] Dans ce cas, nous ne pouvons pas considérer la méthode d'Earley car elle n'inclut pas cette étape de génération de l'analyseur.

considéré est plus simple dans le cas déterministe. Les résultats de ces tests sont présentés dans le premier schéma de la Figure 6 en fonction du concept d'*action élémentaire de construction*, que nous définissons comme étant une action représentant soit l'introduction d'items dans la base ou dans la fermeture de chaque état de l'automate, soit la génération de transitions entre deux états. Cette notion nous permet d'exprimer de façon objective la quantité de travail réalisée par le générateur pour chacun des tests, caractérisés par le nombre de règles de la grammaire définissant le langage, le nombre d'états de l'automate LALR(1) et le nombre de conflits détectés.

Notre but est maintenant de mesurer la qualité de notre approche par rapport aux performances des analyseurs générés. À ce niveau, nous comparons les résultats de trois algorithmes différents: la méthode classique d'Earley, l'algorithme de Tomita [13] et notre adaptation de l'algorithme de Lang [1]. Concrètement, nous considérons les implémentations d'Earley et du schéma LALR(1) proposées par GALENA, et les plus performantes des implémentations de l'algorithme de Tomita à notre connaissance, celles de SDF [22] et GLR [23].

Pour ce faire, nous nous intéressons à un scenario qui ne puisse pas être qualifié comme favorable, quelquesoit le point de vue considéré. À ce point, nous cherchons des grammaires avec les caractéristiques suivantes: le nombre de règles et la taille des textes utilisés doivent être aussi limités que possible, de façon à favoriser la compréhension. Pour la même raison, le langage généré doit être largement connu, et de plus il doit générer des textes avec une proportion importante d'ambigüités afin de prouver l'adéquation du système à cette caractéristique. Les tests doivent inclure un nombre important de mises à jour des textes, pour montrer le niveau d'interactivité entre l'utilisateur et le système, en s'intéressant aux temps de réponse des opérations d'édition. Finalement, la grammaire doit inclure la possibilité de générer un nombre important d'arbres syntaxiques croisés, cas le plus défavorable pour appliquer l'algorithme incrémental.

En relation aux besoins décrits, il semble raisonnable de penser que les langages naturels ne sont pas les plus appropriés pour réaliser ces tests. La grammaire considérée est décrite par l'ensemble de règles suivant:

$$(0)\quad S \rightarrow S\ +\ S \qquad (1)\quad S \rightarrow S\ *\ S \qquad (2)\quad S \rightarrow (\ S\ ) \qquad (3)\quad S \rightarrow \mathit{number}$$

Cette grammaire est de petite taille, elle génère un langage universellement connu, celui des expressions arithmétiques ambigües, et il est très simple d'écrire des textes courts contenant un très grand nombre d'ambigüités et d'arbres croisés, s'ajustant ainsi à toutes les contraintes voulues. Concrètement, nous avons considéré des textes d'entrée de la forme suivante:

$$b\{+b\}^i$$

où i est le nombre de symboles +. Comme la grammaire contient une règle

$$S \rightarrow S\ +\ S$$

les textes considérés demandent un nombre d'analyses qui grandit de façon exponentielle avec i. Les résultats des analyses standard sont présentés dans le deuxième diagramme de la Figure 6.

Afin de fournir des tests pour l'incrémentalité où le nombre de modifications est important, nous changeons des expressions $b + b$ par b dans les exemples précédents. Les résultats de cette étude sont montrés dans la Figure 7.

Tous les tests ont été développés en utilisant les mêmes textes d'entrée pour chacun des analyseurs syntaxiques, et le temps nécessaire pour la sortie en écran des arbres n'a pas été pris en compte.

9 Conclusions

Le système GALENA présenté dans ce travail est une tentative pour offrir un support pour le dévéloppement d'analyseurs de langages naturels, unifiant dans un même environnement la conception du langage et sa validation. Ainsi, le système supporte l'édition simultanée de langages et de textes écrits dans ces langages, en utilisant un mécanisme de description standard sur UNIX pour la définition des grammaires. Une caractéristique importante, en relation à la conception de langages, est la modularité de l'architecture de génération d'analyseurs, permettant à l'utilisateur de considérer des algorithmes spécialisés selon ses besoins, ou de comparer les performances de l'ensemble des méthodes disponibles. Par rapport aux générateurs d'analyseurs les plus

performants, le système proposé non seulement semble améliorer les résultats précédents, mais également il permet le traitement de n'importe quel type de grammaire algébrique indépendamment de sa forme.

De façon à obtenir une efficacité accrue pendant le processus d'analyse, les analyseurs générés par notre système incluent une fonctionnalité incrémentale. Même si l'incrémentalité ne semble pas être un problème trivial, nous avons abouti à le résoudre tout en préservant une complexité raisonnable au niveau du processus. Ainsi, l'analyse de parties de texte peut être entreprise efficacement, sans exiger à l'utilisateur une écriture soignée des textes au risque de devoir répéter l'analyse de la totalité du texte. Des tests pratiques ont prouvé la validité de l'approche proposée lorsque le nombre d'ambigüités reste raisonnable.

Le système inclut une interface graphique qui supporte tout le processus et qui est ouverte à une personnalisation de la part de l'utilisateur. Pour ce faire, chaque composante de GALENA est largement paramétrisée.

Afin de compléter le système, nous travaillons dans l'extension des analyseurs syntaxiques générés dans deux sens: le traitement des grammaires d'unification, essentiel pour l'analyse syntaxique des éléments fonctionnels qui constituent le texte, et la correction automatique des erreurs syntaxiques. De plus, nous intégrerons un module statistique de desambigüation lexicale qui completera les possibilités de desambigüation des analyseurs syntaxiques.

References

[1] M. Vilares Ferro, *Efficient Incremental Parsing for Context-Free Languages*, PhD thesis, University of Nice, France, 1992.

[2] Ch. Donnelly and R.M. Stallman, *BISON. Reference Manual*, Free Software Foundation, Inc., 675 Mass Avenue, Cambridge, MA 02139, U.S.A., 1.20 edition, 1992.

[3] J. Earley, "An efficient context-free parsing algorithm", *Communications of the ACM*, , no. 2, pp. 94–102, 1970.

[4] D.G. Hays, "Automatic language-data processing", in *Computer Applications in the Behavioral Sciences*, H. Borko ed., Ed., pp. 394–423. Prentice-Hall, 1962.

[5] J. Kasami, "An efficient recognition and syntax analysis algorithm for context-free languages", Tech. Rep. AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts, U.S.A., 1965.

[6] D.H. Younger, "Recognition and parsing of context-free languages in time n^3", *Information and Control*, , no. 2, pp. 189–208, 1967.

[7] D. Coopersmith and S. Winograd, "Asymptotic complexity of matrix multiplication", *SIAM Journal on Computing*, , no. 3, pp. 472–492, 1982.

[8] L.G. Valiant, "General context-free recognition in less than cubic time", *Journal of Computer and System Sciences*, pp. 308–315, 1975.

[9] A. Paeseler, "Modification of Earley's algorithm for speech recognition", *NATO ASI Series*, pp. 466–472, 1988.

[10] F.C.N. Pereira and D.H.D. Warren, "Parsing as deduction", in *Proc. of the* 21^{st} *Annual Metting of the Association for Computational Linguistics*, 37-144, Ed., Cambridge, Massachusetts, U.S.A., 1984.

[11] B. Lang, "Deterministic techniques for efficient non-deterministic parsers", Tech. Rep. 72, INRIA, Rocquencourt, France, 1974.

[12] B. Lang, "Towards a uniform formal framework for parsing", in *Current Issues in Parsing Technology*, M. Tomita ed., Ed., pp. 153–171. Kluwer Academic Publishers, 1991.

[13] M. Tomita, "An efficient augmented-context-free parsing algorithm", *Computational Linguistics*, , no. 1–2, pp. 31–36, 1987.

[14] J. Kilbury, "Chart parsing and the Earley's algorithm", Tech. Rep. KIT-Report 24, Projektgruppe Künstliche Intelligenz und Textverstehen, Technische Universität Berlin, West Berlin, Germany, 1985.

[15] M. Tomita, *Efficient Parsing for Natural Language. A Fast Algorithm for Practical Systems*, Kluwer Academic Publishers, Norwell, Massachusetts, U.S.A., 1986.

[16] M. Vilares Ferro, "Efficient sharing in ambiguous parsing", in *Actas del X Congreso de la SEPLN*, Córdoba, España, 1994.

[17] S. Billot and B. Lang, "The structure of shared forest in ambiguous parsing", Tech. Rep. 1038, INRIA, Rocquencourt, France, 1989.

[18] M. Bouckaert, A. Pirotte, and M. Snelling, "Efficient parsing algorithms for general context-free grammars", *Information Sciences*, pp. 1–26, 1975.

[19] M. Vilares Ferro and B. A. Dion, "Efficient incremental parsing for context-free languages", in *Proc. of the* 5^{th} *IEEE International Conference on Computer Languages*, Toulouse, France, 1994, pp. 241–252.

[20] M. A. Alonso Pardo, "Edición interactiva en entornos incrementales", Master's thesis, Computer Sciences Department, University of A Coruña, A Coruña, Spain, 1994.

[21] R.M. Stallman, *GNU Emacs Manual. Version 18*, Free Software Foundation, Inc., 675 Mass Avenue, Cambridge, MA 02139, U.S.A., 1991.

[22] J. Heering, P.R.H. Hendriks, P. Klint, and J. Rekers, "The syntax definition formalism sdf - reference manual", *SIGPLAN Notices*, , no. 11, pp. 43–75, 1989.

[23] J. Rekers, *Parser Generation for Interactive Environments*, PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1992.