Analyse syntaxique automatique de langues : du combinatoire au calculatoire[1]

Jacques Vergne

GREYC - Université de Caen
BP 5186 - 14032 Caen cedex
Jacques.Vergne@info.unicaen.fr

Résumé – Abstract

Nous proposons de montrer comment l'analyse syntaxique automatique est aujourd'hui à un tournant de son évolution, en mettant l'accent sur l'évolution des modèles d'analyse syntaxique : de l'analyse de langages de programmation (compilation) à l'analyse de langues, et, dans le cadre de l'analyse de langues, de l'analyse combinatoire à l'analyse calculatoire, en passant par le tagging et le chunking (synthèse en section 4). On marquera d'abord le poids historique des grammaires formelles, comme outil de modélisation des langues **et** des langages formels (section 1), et comment la compilation a été transposée en traduction automatique par Bernard Vauquois. On analysera ensuite pourquoi il n'a pas été possible d'obtenir en analyse de langue un fonctionnement analogue à la compilation, et pourquoi la complexité linéaire de la compilation n'a pas pu être transposée en analyse syntaxique (section 2). Les codes analysés étant fondamentalement différents, et le tagging ayant montré la voie, nous en avons pris acte en abandonnant la compilation transposée : plus de dictionnaire exhaustif en entrée, plus de grammaire formelle pour modéliser les structures linguistiques (section 3). Nous montrerons comment, dans nos analyseurs, nous avons implémenté une solution calculatoire, de complexité linéaire (section 5). Nous conclurons (section 6) en pointant quelques évolutions des tâches de l'analyse syntaxique.

In this paper, we intend to show how automatic parsing is today at a change of direction. We will stress the evolution of parsing models : from programming language parsing (compilation) to natural language parsing, and, within the frame of natural language parsing, from combinatory parsing to calculatory parsing, while going through tagging and chunking (synthesis in section 4). First we stress the historical weight of formal grammars, as a modelling tool for natural languages **and** formal languages (section 1), and how compilation has been transposed into machine translation by Bernard Vauquois. Then we analyse why it was not possible to get a natural language parsing which works the same way as compilation, and why the linear complexity of compilation could not be transposed into NL parsing

---

[1] Cet article consiste en une synthèse du contenu de ma conférence invitée à TALN'2001. La présentation est disponible sur : `http://www.info.unicaen.fr/~jvergne/TALN2001_JV.ppt`

(section 2). Since parsed languages are radically different, and since tagging showed the right way, we decided to abandon the transposed compilation : no more exhaustive dictionary as input, no more formal grammar to model linguistic structures (section 3). We will show how, in our parsers, we implemented a calculatory solution, of linear complexity (section 5). We conclude (section 6), stressing some trends of parsing about its tasks.

Introduction

Nous proposons une étude des modèles d'analyse syntaxique : origines, évolution, analogies, ruptures, et continuités. Les critères sont : le modèle du code analysé, la place de ce modèle dans l'analyseur, le processus d'analyse et sa complexité.

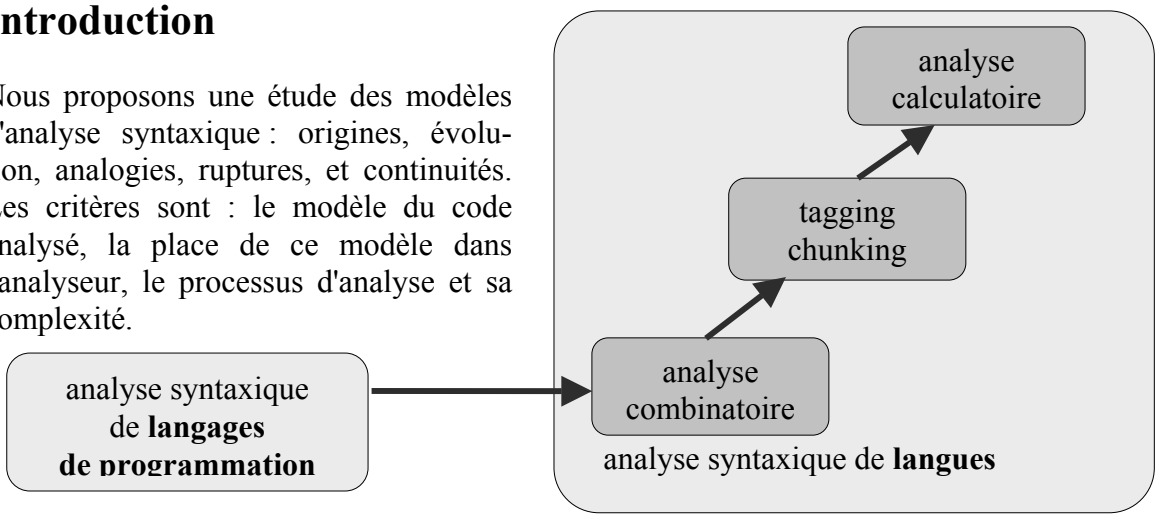

Figure 1 : Plan général

1 Le poids historique des grammaires formelles

Les grammaires formelles permettent de modéliser des objets différents dans les trois domaines, linguistique, informatique et TAL, ce qui entraîne une certaine confusion sur leur place en TAL :

1.1 Les grammaires formelles, outil de modélisation de la syntaxe des langues

Noam Chomsky a eu une formation initiale de mathématicien, puis il est entré en linguistique, dans la filiation de Harris et Bloomfield, mais en rupture avec eux sur l'objet étudié : Harris et Bloomfield étudiaient le matériau attesté, alors que Chomsky a estimé impossible d'induire une théorie à partir d'un matériau attesté nécessairement fini, ce qui l'a conduit à définir son objet comme étant la "compétence" du locuteur natif, concept qu'il a lui-même élaboré (Chomsky, 1965).

En 1957, il publie *Syntactic Structures*, un ouvrage de linguiste (Chomsky, 1957).

Pour les linguistes chomskiens, les grammaires formelles servent à modéliser la compétence du locuteur natif, en génération (considérée comme la déduction de la démarche hypothético-déductive). Mais, pour la communauté des Traitements Automatiques des Langues, elles servent à modéliser la syntaxe des langues, comme matériau attesté, en analyse.

La divergence entre les deux courants s'est produite vers 1971, avec la Théorie Standard Étendue, présente dans les formalismes syntaxiques classiques en TAL (Abeillé, 1993).

1.2 Les grammaires formelles, outil de modélisation de la syntaxe des langages de programmation

Certains concepts élaborés par Chomsky dès 1957 trouvent une application inattendue en 1960, avec la création d'ALGOL 60, premier langage de programmation dont la syntaxe est définie par une grammaire formelle (context free), ce qui guide la conception du compilateur.

ALGOL est le premier Langage de programmation Orienté "ALGorithmique" : il n'y a plus de `goto` obligatoire dans les structures de contrôle, mais les structures *alternative* (`if ... then ... else`) et *répétitive* (`for ... step ... until ... do` et `for ... while ... do`).

ALGOL 60 est aussi le premier langage à structure de **bloc récursif**, définie par la grammaire formelle suivante :

programme —> instruction_complexe *
instruction complexe —> instruction_simple | **bloc**
bloc —> instruction_complexe *

ALGOL 60 aura eu une filiation prolifique : Pascal, C, Ada, C^{++}, et Java.

1.3 Triple présence des grammaires formelles en TAL

Les Traitements Automatiques des Langues sont à l'intersection de la linguistique et de l'informatique et héritent d'une partie de leurs concepts et de leur culture. Les grammaires formelles se retrouvent donc dans trois modèles différents dans les trois disciplines :

- en linguistique, le modèle de la "compétence" du locuteur natif, en génération
- en informatique, le modèle de la syntaxe des langages de programmation
- en TAL, le modèle de la syntaxe des langues (matériau attesté), en analyse.

Cette triple présence conduit à une situation confuse des grammaires formelles en TAL.

2 De l'analyse de langages de programmation à l'analyse de langues

Bernard Vauquois fut le messager qui a transposé les grammaires formelles de l'informatique vers la Traduction Automatique :

2.1 Bernard Vauquois, membre du groupe ALGOL

Sept pays ont participé à la conception d'ALGOL : Allemagne, Danemark, États-Unis, France, Grande-Bretagne, Pays-Bas, et Suisse. Les conférences se sont échelonnées dès 1958

: Zurich (1958), Mayenne (1958), Copenhague (1959), Paris (juin 1959, janvier 1960). Quatorze délégués y participèrent : Backus, Bauer, Green, Katz, Mc Carthy, Naur, Perlis, Rutishauser, Salmelson, Turanski, **Vauquois**, Wegstein, Van Wijngaarden, Woodger.

2.2 Bernard Vauquois, directeur du CETA en 1961

Bernard Vauquois fut d'abord astronome et mathématicien, puis enseignant en informatique, en théorie des langages formels à l'université de Grenoble. En 1961, alors qu'il vient de participer à la création d'ALGOL, il prend la direction du CETA (Centre d'Études pour la Traduction Automatique) à Grenoble. Il fonde alors la Traduction Automatique (TA) de 2ème génération sur les principes explicites suivants :

- utiliser la théorie des langages formels
- fonder la Traduction Automatique sur le modèle de la compilation (Boitet, 1989).

2.3 Transposition explicite de la compilation en TA par Bernard Vauquois

Nous pouvons concevoir la programmation comme la traduction par un humain d'instructions pensées et exprimées en une **langue**, en instructions exprimées en **langage machine** destinées à être exécutées par un processeur; cette opération de traduction consiste en deux étapes : une étape de traduction humaine (appelée analyse-programmation) qui conduit à l'écriture des instructions en un **langage de programmation**, suivie d'une étape de traduction automatique, appelée compilation. La compilation est une traduction automatique entre deux **langages formels**, donc clos et totalement connus, ce qui permet justement d'automatiser la traduction.

La traduction automatique de langue à langue, deux codes ouverts, connus partiellement, est pour ces raisons de nature radicalement différente de la compilation.

Or, Bernard Vauquois transpose explicitement la compilation dans la traduction automatique de langues. Voici ce qu'écrit Christian Boitet, page 3 de l'introduction des Analectes, "L'apport scientifique de Bernard Vauquois" (Boitet, 1989) :

Au tout début de la TA, on fit l'analogie entre traduction et décodage (Warren Weaver, 1949), pour se rendre assez vite compte de son manque de pertinence. Il revient sans doute au CETA, à l'initiative de B. Vauquois, d'avoir introduit l'analogie beaucoup plus féconde entre TA et compilation. [...], on cherche seulement à transformer la forme en préservant le sens, de même qu'un compilateur transforme un programme en un programme équivalent, [...]

Ainsi un système de TA est-il vu comme une sorte de "compilateur de langue naturelle".

2.4 De la compilation à l'analyse de langues

La transposition de la compilation reste aujourd'hui valide dans l'analyse de langues dans son mode classique (et combinatoire). On notera en particulier que l'on a en entrée le même modèle du code analysé, mais que les processus sont différents :

- même modèle du code analysé : ressources lexicales (dictionnaire) et syntaxiques (grammaire formelle) exhaustives placées en entrée

- mais processus différents :

critères	*compilation*	*analyse de langues*
processus	répétitif / token déterministe	combinatoire **non** déterministe
complexité en temps	*théorique* : polynomiale *pratique* : linéaire	*théorique* : exponentielle *pratique* : polynomiale

Tableau 2 : De la compilation à l'analyse de langue : comparaison des processus

Les processus sont différents car les codes analysés sont différents : langage de programmation versus langue.

2.5 Quelle différence entre les deux codes analysés ?

La question qui se pose est alors : quelle est la différence entre les codes analysés qui provoque le caractère combinatoire de l'analyse de langues ?

critères	*langages de programmation*	*langues*
dictionnaire	énuméré, **fermé** et figé	**ouvert** et évolutif
combien d'étiquettes par token ?	1 étiquette **unique**	**plusieurs** étiquettes

Tableau 3 : Comparaison entre langages de programmation et langues

Dans une langue, un token a **plusieurs** étiquettes possibles : c'est la cause principale du caractère combinatoire de l'analyse de langue (dans son mode classique)[2].

3 Analyse de langues : du combinatoire au calculatoire

Dans cette section, nous rappelons par un exemple ces deux modes de résolution d'un problème, et nous caractérisons plus précisément comment poser et résoudre un problème de manière combinatoire. Puis nous montrons comment construire une résolution calculatoire. Nous passons ensuite au cas particulier de l'analyse syntaxique de langue : la poser et la résoudre de manière combinatoire, puis calculatoire. Puis nous montrons le rôle déterminant du tagging, comme passerelle vers l'analyse de langue calculatoire. Enfin nous présentons les grandes lignes de l'analyse de langue calculatoire.

---

[2] Les autres causes de combinatoire sont à rechercher dans les autres "plusieurs" du processus d'analyse : principalement, **plusieurs règles** sont applicables au même moment.

3.1 Un exemple des deux modes de résolution

- Voici un problème :

  un fils a le quart de l'âge de son père, et leur différence d'âge est de 30 ans

  $f = p / 4$ $\qquad\qquad$ $p - f = 30$

- Voici sa résolution combinatoire :

  avec $0<f<100$, $0<p<100$ (en supposant une solution entière) :
  pour chacun des 10 000 couples (f, p),
  tester les 2 contraintes;
  le nombre de solutions est inconnu a priori.

- Et voici sa résolution calculatoire :

  résoudre le système des 2 équations à 2 inconnues, ce qui implique une solution unique :

  $f = p/4$ et $p-f=30$ => $4f-f = 3f = 30$ => $f=10$ => $p=30+f=40$

3.2 Poser et résoudre un problème : du combinatoire au calculatoire

Le terme "combinatoire" dénote une manière de **poser** et **résoudre** un problème, mais non pas le problème lui-même.

Voici comment on **pose** un problème de manière combinatoire : les attributs d'un ensemble d'unités ont plusieurs valeurs possibles, on a des contraintes sur les valeurs des attributs, et on veut trouver les valeurs des attributs qui satisfont les contraintes. Ceci revient à poser le problème comme un **problème de satisfaction de contraintes** (CSP).

Dans un processus de **résolution** combinatoire, toutes les combinaisons de valeurs sont constituées incrémentalement, dans un parcours d'arbre en profondeur d'abord (en analyse syntaxique combinatoire, la profondeur correspond au nombre de mots, et l'arité d'un nœud correspond au nombre d'étiquettes possibles d'un mot). Chaque combinaison partielle ou complète satisfait ou non les contraintes. Si les contraintes ne sont pas satisfaites, il y a un retour en arrière au dernier nœud satisfaisant les contraintes, et un essai d'une nouvelle valeur possible. La complexité théorique en temps est exponentielle selon le nombre de mots.

Dans une résolution **combinatoire**, on a trouvé comment vérifier la satisfaction des contraintes sur une combinaison *déjà constituée*: la combinaison est en **entrée** de la **validation** par les contraintes. Le résultat est déjà présent, énuméré, explicité, mais caché et dispersé dans les ressources, et il faut l'en extraire.

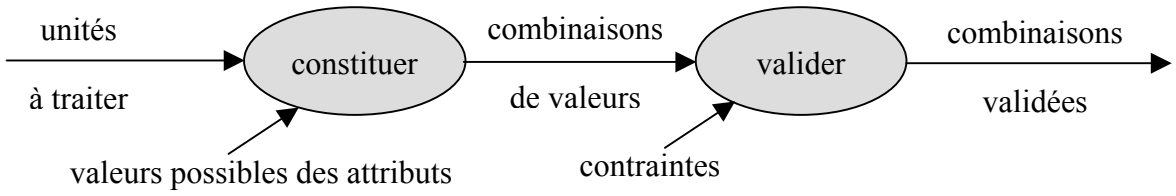

Figure 4 : Résolution combinatoire d'un problème

Pour construire une résolution **calculatoire**, il faut avoir une meilleure connaissance des propriétés des unités à traiter et mieux exploiter cette connaissance, et trouver plus de contraintes et la manière de les utiliser pour calculer directement et définitivement la valeur d'un attribut : la "combinaison" est alors en **sortie** du **calcul** par les contraintes :

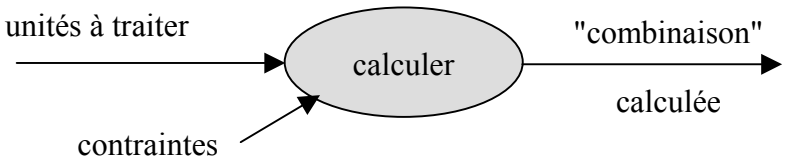

Figure 5 : Résolution calculatoire d'un problème (calcul direct)

On peut aussi dire que, dans une résolution calculatoire, il faut trouver des **opérations** (en petit nombre, placées en ressources et exécutées par l'algorithme) portant sur des **opérandes** (les unités à traiter placées en entrée). Ce qui était "contraintes" est alors vu comme des opérations sur les unités à traiter :

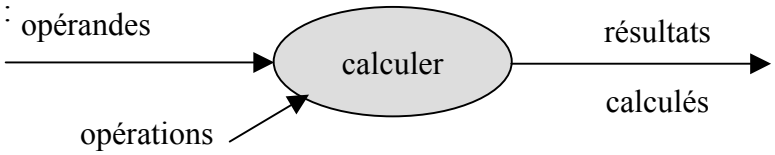

Figure 6 : Résolution calculatoire d'un problème : opérations sur des opérandes

3.3 Poser et résoudre l'analyse syntaxique de manière combinatoire

Le problème de l'analyse syntaxique automatique est traditionnellement **posé** et **résolu** de manière combinatoire.

Le problème est ainsi **posé** :

- les mots d'une phrase ont plusieurs catégories possibles,
- toutes les valeurs possibles des attributs des mots d'une phrase sont explicitées "exhaustivement" dans le *dictionnaire* (toutes les graphies, catégories, genres, nombres, personnes, ... des mots),
- les contraintes sur les catégories sont les structures possibles des phrases et des syntagmes, explicitées "exhaustivement" dans la *grammaire formelle*,
- et, pour les mots de la phrase entrante, on veut trouver les catégories qui satisfont les contraintes (= "désambiguïsation").

Résolution : à partir des mots de la phrase entrante et du dictionnaire, un processus combinatoire **constitue** des combinaisons des catégories possibles des premiers mots. Chaque combinaison partielle est **validée** de manière booléenne sur la grammaticalité du début de la phrase, selon la grammaire placée en entrée. Une phrase est analysée entièrement si on trouve une combinaison sur la phrase, ce qui correspond à la grammaticalité de la phrase entière. Le

nombre de solutions est inconnu a priori : 0, 1 ou beaucoup, sans critères de choix entre solutions.

3.4 Analyse combinatoire —> tagging —> analyse calculatoire

Le tableau suivant montre :

- la rupture entre analyse combinatoire et tagging,
- et la continuité entre tagging et analyse calculatoire :

critères	*analyse de langues combinatoire*	*tagging, chunking*	*analyse de langues calculatoire*
structures attendues	explicitées sous forme d'une grammaire formelle	pas de structure attendue	pas de structure attendue
ressources syntaxiques	exhaustives (grammaire formelle)	partielles (règles contextuelles)	règles : condition => action
ressources lexicales	exhaustives (dictionnaire)	exhaustives (ou partielles)	mots grammaticaux seulement
processus	arborescent, combinatoire, non déterministe	répétitif par token, déterministe	répétitif par token, déterministe
complexité en temps	*théorique* : exponentielle *pratique* : polynomiale	*théorique* : linéaire, *pratique* : linéaire	*théorique* : linéaire, *pratique* : linéaire
code analysé	langue	langue	langue

Tableau 7 : De l'analyse de langues combinatoire à l'analyse de langues calculatoire, en passant par le tagging et le chunking.

Le tagging amorce une évolution fondamentale en analyse linguistique :

- l'abandon du modèle des structures **en entrée**,
- l'abandon des grammaires formelles pour modéliser les structures,
- et l'utilisation explicite du contexte

permettent de construire un processus de complexité linéaire.

Le tagging dans sa version classique utilise, comme l'analyse combinatoire, des ressources lexicales supposées exhaustives, si l'on considère qu'il s'agit de **réduire** les listes exhaustives d'étiquettes possibles extraites du dictionnaire, par l'application des règles contextuelles. Mais il est aussi possible de tagger avec des ressources partielles : les mots grammaticaux (avec une seule étiquette par défaut) qui marquent le début d'un chunk (Abney, 1991) et qui le typent nominal ou verbal, type qui contraint les mots suivants du chunk (Vergne, 1998b).

Le chunking fonctionne exactement comme le tagging, et simultanément au tagging, les deux fonctions étant simplifiées quand on les aborde ensemble (voir ci-dessous en 5.1).

Tagging et chunking ouvrent et montrent la voie vers l'analyse de langue calculatoire : les fonctions de segmentation et d'identification des segments sont réalisées; il suffit d'y ajouter un processus de mise en relation des segments qui soit aussi implémentable par des règles conditions => actions[3] (voir ci-dessous en 5.2).

3.5 Poser et résoudre l'analyse syntaxique de manière calculatoire

L'**ouverture** de la langue est prise en compte explicitement : une langue n'est pas caractérisable exhaustivement, contrairement à un langage de programmation; les ressources lexicales sont minimales : mots grammaticaux, morphèmes de fin de mots; il n'y a pas de grammaire formelle, c'est-à-dire pas d'inventaire exhaustif des structures attendues, et pas de test de grammaticalité.

Le modèle linguistique est dissocié du modèle de traitement informatique, et n'est pas **implémenté** tel quel dans sa totalité (dictionnaire, grammaire), mais il est partiellement et implicitement **utilisé** dans la rédaction des règles de calcul.

Le processus de **calcul** est exécuté par un moteur à base de règles, et consiste à passer des *règles* "conditions => actions" une fois sur chaque unité : caractères, tokens, syntagmes, propositions, phrases, (paragraphes, ..., textes entiers). Les *conditions* portent sur les attributs d'unités et sur les relations entre unités (contiguïtés, constituances et dépendances). Les *actions* consistent à affecter des valeurs aux attributs, établir des relations, et générer les unités du niveau supérieur. Le traitement est en flux à débit constant, sans découper a priori une unité à traiter dans sa totalité telle que la phrase. Un élément du flux est traité complètement, une fois pour toutes, en passe unique, avant de passer à l'élément suivant.

Les règles explicitent dans un même formalisme :

- les ressources lexicales et morphologiques partielles,
- les propriétés contextuelles (utilisation explicite du contexte),
- et le processus de mise en relation (voir ci-dessous en 5.2).

Le lexique et les structures syntaxiques du texte analysé sont calculés et produits en sortie. Le processus d'analyse est de complexité linéaire.

On a ainsi une nouvelle manière de modéliser les propriétés linguistiques des langues dans les traitements informatiques, qui consiste à ne transposer dans le modèle informatique qu'un tout petit nombre de propriétés générales du modèle linguistique, seulement celles qui servent aux

---

[3] voir aussi la présentation et les références du tutoriel du Coling 2000 "Trends in Robust Parsing" sur `http://www.info.unicaen.fr/~jvergne/tutorialColing2000.html` (Vergne, 2000).

opérations des calculs. Le modèle linguistique n'est plus implémenté dans sa totalité. Il est ainsi dissocié du modèle informatique[4].

4 Comparaison entre modèles d'analyse syntaxique

Le tableau suivant propose une synthèse comparative entre les quatre modèles d'analyse et met en évidence :

- les ruptures et les continuités entre modèles d'analyse syntaxique,
- le retour à la complexité linéaire du processus d'analyse,
- et le fait que le modèle du code analysé n'est plus implémenté dans sa totalité dans le modèle informatique sous forme de ressources exhaustives placées en entrée. Seules quelques propriétés générales du code analysé servent à définir les opérations du calcul.

critères	*compilation*	*analyse de langues combinatoire*	*tagging, chunking*	*analyse de langues calculatoire*
structures attendues	grammaire formelle	grammaire formelle	pas de structure attendue	pas de structure attendue
ressources syntaxiques	exhaustives (grammaire formelle)	exhaustives (grammaire formelle)	partielles (règles contextuelles)	règles : condition => action
ressources lexicales	exhaustives (primitives)	exhaustives (dictionnaire)	exhaustives ou partielles	mots grammaticaux seulement
processus	répétitif / token, déterministe	arborescent, combinatoire, non déterministe	répétitif / token, calculatoire, déterministe	répétitif / token, calculatoire, déterministe
complexité en temps	*théorique* : polynomiale *pratique* : linéaire	*théorique* : exponentielle *pratique* : polynomiale	*théorique* : linéaire, *pratique* : linéaire	*théorique* : linéaire, *pratique* : linéaire
code analysé	langage formel	langue	langue	langue

Tableau 8 : Comparaison entre les quatre modèles d'analyse syntaxique

[4] Le thème de la confusion des modèles linguistique et informatique sera développé par Thomas Lebarbé dans sa thèse de doctorat.

5 Quelques caractéristiques de nos analyseurs

Nous proposons de mettre l'accent sur quelques principes fondamentaux de nos analyseurs : le calcul forme-position, le processus de mise en relation des unités, et les méthodes pour s'abstraire du niveau d'unité traitée, et de la langue traitée.

"Nos analyseurs" s'entend ainsi :

- "l'analyseur 98" (conçu et réalisé en solitaire durant et depuis ma thèse soutenue en 1989 : "Analyse morpho-syntaxique automatique sans dictionnaire"), est non combinatoire depuis 1993; c'est un analyseur de phrases du français; il s'est placé premier à l'action GRACE (d'octobre 1995 à novembre 1998), et a été transposé dans la synthèse vocale KALI (Vannier, 1999), commercialisée depuis 1999 par Electrel (financement FEDER);
- "l'analyseur du GREYC" a été conçu (à partir du premier) et réalisé en 1999-2000, en collaboration avec Emmanuel Giguet (Giguet, 1998) et Nadine Lucas (membres du Groupe Syntaxe et Ingénierie Multilingue du GREYC), dans le cadre d'un transfert de technologie vers la société DATOPS, (financement MENRT, contrat n° 98.K.6411), et il est actuellement en service dans les traitements de l'entreprise; c'est un analyseur linguistique générique : les règles actuellement développées permettent l'analyse morpho-syntaxique de phrases du français et de l'anglais et la détection des citations (Lucas, 2000b). Il a été développé en Java; le moteur compilé et les règles tiennent sur moins de 500 Ko.

5.1 Calculer à partir des formes *et* des positions

On abandonne la correspondance formes - étiquettes possibles a priori hors contexte habituellement explicitée dans le dictionnaire exhaustif. Les formes connues au début du calcul sont les mots grammaticaux qui marquent le début d'un chunk et qui le typent nominal ou verbal, type qui contraint les mots suivants du chunk (Vergne, 1998b) :

Je *le bois* ***dans*** *le bois.* (exemple explicatif)

- début et typage de chunk : **Je** => [chunk verbal **dans** => [chunk nominal

  *[**Je** le bois]* *[**dans** le bois]* .
  chunk verbal chunk nominal

- contraintes du type de chunk sur l'étiquette des mots :

  *[**Je*** *le* *bois]* *[**dans*** *le* *bois]* .
  pronom verbe déterminant nom

Le calcul porte ainsi sur les formes **et** les positions, et produit la segmentation et l'étiquetage des segments.

5.2 Un processus de complexité linéaire pour relier les unités

Soit le flux des unités (chunks par exemple) traitées au fur et à mesure de leur arrivée par le moteur. Les conditions des règles portent sur l'unité courante et sur toute unité qui lui est reliée (relations de contiguïté, constituance et dépendance). Soit une unité virtuelle invocable à tout moment dans les conditions des règles, et servant d'intermédiaire dans la mise en relation par un processus en deux temps :

- au temps 1, à l'arrivée de l'unité i, une règle 1 pose que cette unité attend une éventuelle unité j en la reliant à l'unité virtuelle par un lien typé;
- au temps 2, à l'arrivée de l'unité j, une règle 2 examine si une unité i attend une unité j en consultant les liens typés de l'unité virtuelle à une unité i, puis supprime le lien entre l'unité i, relie l'unité i à l'unité j par un lien du même type, et enfin supprime tous les liens à l'unité virtuelle de toutes les unités situées entre les unités i et j (ces attentes sont résolues et supprimées)[5].

Les attentes non résolues à la fin d'unités supérieures (phrases par exemple) sont supprimées par une règle 3. Ce processus est implémenté dans les règles interprétées par le moteur, et ainsi est aussi de complexité linéaire. Il est indépendant des unités arrivées entre les deux unités reliées.

Du point de vue linguistique, ce processus modélise le déroulement d'une saturation de valence.

5.3 Comment s'abstraire de l'unité traitée ?

Les opérations sont identiques quel que soit le niveau hiérarchique des unités traitées : par exemple, les débuts de propositions sont marqués de manière analogue aux débuts de chunks (Lucas, 2000a). Les deux hiérarchies des unités sont déclaratives : on a deux hiérarchies de constituants non récursifs en miroir :

- hiérarchie des unités physiques : document, paragraphes, mots
- hiérarchie des unités calculées : mots, tokens, chunks, propositions, phrases

5.4 Comment s'abstraire de la langue ?

Certaines propriétés du flux entrant sont indépendantes de sa langue :

- le flux entrant est toujours unidimensionnel
- ce sont toujours deux humains qui communiquent (c'est un processus) : même appareil vocal, même système cognitif, même recherche du moindre effort (optimisation).

Le chunk est un constituant fondé sur l'oral (un groupe accentuel), contraint par l'appareil vocal (Abney, 1991). Nous faisons l'hypothèse que c'est un concept indépendant de la langue,

---

[5] On pourra se reporter à l'animation de la présentation pour faciliter la compréhension du processus.

hypothèse corroborée par les travaux de Hervé Déjean (Déjean, 1998) sur des corpus de langues très variées.

Dans l'analyse automatique, comment s'abstraire de la langue du flux entrant ? Un premier paquet de règles affecte des valeurs à des attributs de certains mots à partir de leur graphie (une unique valeur par défaut, valeur initiale des calculs). Les paquets suivants provoquent des calculs sur les attributs, calculs indépendants de la langue du flux.

Dans l'analyseur du GREYC, analyseur syntaxique de l'anglais et du français, les opérations communes anglais - français sont les suivantes : segmentation en propositions, mise en relation des chunks dans les propositions, segmentation en phrases. Les règles sont communes pour les deux langues et sont mises au point sur corpus anglais - français.

6 Évolution des tâches de l'analyse syntaxique

Le domaine opératoire des analyseurs combinatoires correspond aux applications qui traitent des phrases courtes (requêtes des pages jaunes par exemple), soit où une complexité non linéaire est supportable (la TA en traitement par lot par exemple). Le domaine opératoire des analyseurs calculatoires concerne des applications où le traitement à débit constant est requis : les traitements en temps réel comme la synthèse vocale, les traitements en flux à haut débit, comme l'indexation de dépêches sur internet pour la recherche d'informations.

On peut s'interroger sur la prégnance des traditions scolaires : les mots dans la phrase, les parties du discours. Quel est le rapport avec la tâche ? D'autres grains (document entier, paragraphe, chunk), d'autres catégorisations (chunk nominal ou verbal par exemple) sont possibles.

Pourquoi imiter l'humain ? Par exemple, dans l'évaluation du tagging, la référence est l'imitation de l'humain étiqueteur; dans quel but ?

Au sujet des modèles linguistiques, il n'est pas nécessaire de les **implémenter**. Les études linguistiques sont faites en amont, mais elles ne sont inscrites que **partiellement** et **implicitement** dans les systèmes.

La tâche conditionne les traitements, et l'analyse syntaxique n'est pas toujours un premier traitement obligatoire. On cherchera plutôt à concevoir des outils légers et rapides (complexité linéaire obligatoire), éventuellement interactifs, utilisant des ressources restreintes, ce qui permet des traitements peu dépendants de la variété des langues.

La frontière syntaxe - sémantique s'efface progressivement dans les traitements : on fait des traitements sur les formes, en sélectionnant des documents, en coloriant des passages par un calcul portant sur quelques marques et leurs positions relatives entre elles et dans les segments (début, milieu, fin de segment), et c'est ensuite à l'humain utilisateur d'interpréter les résultats.

Pour les applications de recherche d'informations, les fonctions des termes deviennent prioritaires par rapport aux étiquettes de catégorie qui ne sont que des intermédiaires de calcul partiellement résolus. Les intérêts des utilisateurs évoluent vers les entités nommées, les prises de parole, les thèmes ou la tonalité des documents. On s'oriente vers l'analyse de

documents entiers : le grain-mot devient trop fin et secondaire. Dans un flux massif de documents, un document n'est plus lui-même qu'un grain minuscule.

Conclusion

L'analyse de langue combinatoire est issue de la transposition de la compilation, mais seul le modèle du code analysé est transposé, et, du fait que les mots d'une langue ont plusieurs étiquettes possibles dans le dictionnaire, la faible complexité de la compilation n'est pas au rendez-vous : le problème est posé et résolu comme un problème de satisfaction de contraintes, d'où sa complexité théorique exponentielle. L'analyse de langue combinatoire utilise comme ressources toutes les formes possibles des mots, des syntagmes et des phrases sous la forme d'un dictionnaire et d'une grammaire formelle.

L'abandon de ces ressources exhaustives, associé à des calculs à partir de formes partielles et de leurs positions, a permis de rendre l'analyse de langue calculatoire. Ces calculs n'utilisent que quelques propriétés du modèle linguistique, qui est ainsi dissocié du modèle informatique.

La rupture entre les deux paradigmes combinatoire et calculatoire s'est produite avec l'arrivée du tagging. Est-ce une rupture de paradigme à la Kuhn (Kuhn, 1983) ?

Références

Abeillé A. (1993), *Les nouvelles syntaxes*, Paris, Colin.

Boitet Ch. (1989), *Bernard Vauquois et la TAO - Analectes*, édité par Christian Boitet, GETA, Grenoble.

Abney S. (1991), Parsing by Chunks, In : Robert Berwick, Steven Abney and Carol Tenny (eds.), *Principle-Based Parsing*, Dordrecht, Kluwer Academic Publishers, téléchargeable sur : `http://www.sfs.nphil.uni-tuebingen.de/~abney/Abney_90e.ps.gz`

Chomsky N. (1969), *Structures syntaxiques*, Paris, Point Seuil, trad. fr. de : (1957), *Syntactic structures*, La Haye, Mouton & Co.

Chomsky N. (1971), *Aspects de la théorie syntaxique*, Paris, Seuil, trad. fr. par J.-C. Milner de : (1965), *Aspects of the Theory of Syntax*, Cambridge, MIT Press.

Déjean H. (1998), *Concepts et algorithmes pour la découverte des structures formelles des langues*, thèse de doctorat de l'Université de Caen.

Giguet E. (1998), *Méthode pour l'analyse automatique de structures formelle sur documents multilingues*, thèse de doctorat de l'Université de Caen.

Kuhn Th. (1983), *La structure des révolutions scientifiques*, Paris, Flammarion, trad. fr. de : (1962-1970), *The Structure of Scientific Revolutions*, Chicago, The University of Chicago Press.

Lucas N. (2000a), Le changement d'échelle dans l'analyse logique, *9èmes Rencontres interdisciplinaires sur les systèmes complexes naturels et artificiels : Représentations graphiques dans les systèmes complexes naturels et artificiels*, Rochebrune.

Lucas N. (2000b), Le rôle de la citation dans la structuration des articles de presse, Actes du *premier colloque d'études japonaises de l'Université Marc Bloch*, Strasbourg.

Vannier G. (1999), *Étude des contributions des structures textuelles et syntaxiques pour la prosodie : application à un système de synthèse vocale à partir du texte*, thèse de doctorat de l'Université de Caen.

Vergne J. (1989), *Analyse morpho-syntaxique automatique sans dictionnaire*, thèse de doctorat de l'Université Paris 6.

Vergne J. (1998a), Entre arbre de dépendance et ordre linéaire, les deux processus de transformation : linéarisation, puis reconstruction de l'arbre, *Cahiers de Grammaire*, n° 23, pp. 95-136, ERSS, Toulouse**.**

Vergne J., Giguet E. (1998b), Regards Théoriques sur le «Tagging», Actes de *Traitement Automatique des Langues Naturelles (TALN'98)*, pp. 22-31, Paris.

Vergne J. (1999), *Étude et modélisation de la syntaxe des langues à l'aide de l'ordinateur, Analyse syntaxique automatique non combinatoire*, Habilitation à Diriger des Recherches, Université de Caen.

Vergne J. (2000), *Trends in Robust Parsing*, tutoriel du Coling 2000, téléchargeable sur
`http://www.info.unicaen.fr/~jvergne/tutorialColing2000.html`