Analyse Syntaxique du Japonais

Mémoire de DEA

Yayoi NAKAMURA-DELLOYE
Institut National des Langues et Civilisations Orientales

17 juillet 2003

亡き父、中村一に捧ぐ
À la mémoire de mon père, Hajime NAKAMURA,
qui nous a quitté le 02 avril 2003.

Je tiens à remercier M^{me} Marie-Anne Moreaux et M. Michel Fanton pour leurs encouragements tout au long de la rédaction de ce mémoire, ainsi que pour tous les nombreux conseils qu'ils ont eu la gentillesse de prodiguer.

Je remercie également mes fils, Guy et Noé, qui ont dû supporter, dès leur première année de vie, une jeune maman plus souvent devant l'ordinateur que devant eux.

Enfin, je remercie tout particulièrement mon mari, Olivier, pour son soutien aussi bien sur le plan pratique (corrections orthographiques, installation de LaTeX, etc.) que sur le plan moral, et sans lequel ce mémoire n'aurait sûrement jamais été terminé.

Table des matières

Introduction

Les progrès considérables des équipements informatiques et des télécommunications nous ont apporté une très grande liberté d'accès sur le plan de l'information. Mais nous ne sommes pas encore tout à fait débarrassés de cette grande barrière, dressée devant nous de manière à nous empêcher d'atteindre des endroits encore inconnus, qu'est la langue.

Le Traitement Automatique des Langues (ci-après TAL) est un domaine dans lequel les chercheurs se battaient et se battent encore avec cet immense obstacle, dans le cadre de l'informatique. Ce champ relativement nouveau, créé par la confrontation de deux disciplines a priori très différentes, l'informatique et la linguistique, couvre la totalité des traitements par des moyens informatiques nécessitant des connaissances linguistiques. Comme le dit Jean-Marie PIERREL dans l'introduction de [13] :

> Les finalités de ce nouveau champ disciplinaire, le traitement automatique des langues (TAL), sont multiples. [...] Tout d'abord, elles correspondent à la mise en place d'applications concrètes (indexation et accès à l'information, résumé de textes, traduction assistée par ordinateur, dialogue homme-machine par exemple).

Dans le domaine du TAL, le Japon a été très en avance du fait de la particularité de son système d'écriture et du fort développement économique de l'archipel : il a été très tôt nécessaire de mettre au point des systèmes de traitement permettant la saisie des caractères japonais spécifiques (uniquement à l'aide d'informations phonétiques) dans un environnement conçu initialement pour les alphabets occidentaux. Ainsi, de nombreuses applications ont été développées au Japon, y compris des analyseurs morphologiques et syntaxiques.

Nous nous intéressons dans le présent mémoire à l'analyse syntaxique automatique du japonais. L'analyse syntaxique constitue, avec l'analyse morphologique, une des premières étapes essentielles de toute application d'analyse automatique des langues.

Dans le domaine du TAL, l'analyse automatique des langues est décomposée en sous-tâches, suivant le principe des modèles informatiques qui dé-

composent une procédure complexe en une série de tâches simples, chacune se comportant de manière indépendante des autres. Ces sous-tâches sont calquées sur les différents niveaux de structure linguistique.

Dans le domaine de la linguistique, cette décomposition en plusieurs niveaux a été réalisée suite au constat des linguistes, qui avaient découvert l'existence d'une série de niveaux communs à toute langue. À ces différents niveaux, correspondaient différentes unités linguistiques élémentaires et traditionnelles, comme par exemple, les sons, les mots et les phrases, ce qui a facilité cette découverte.

La figure 1, issu de [56], représente un modèle de compréhension linguistique.

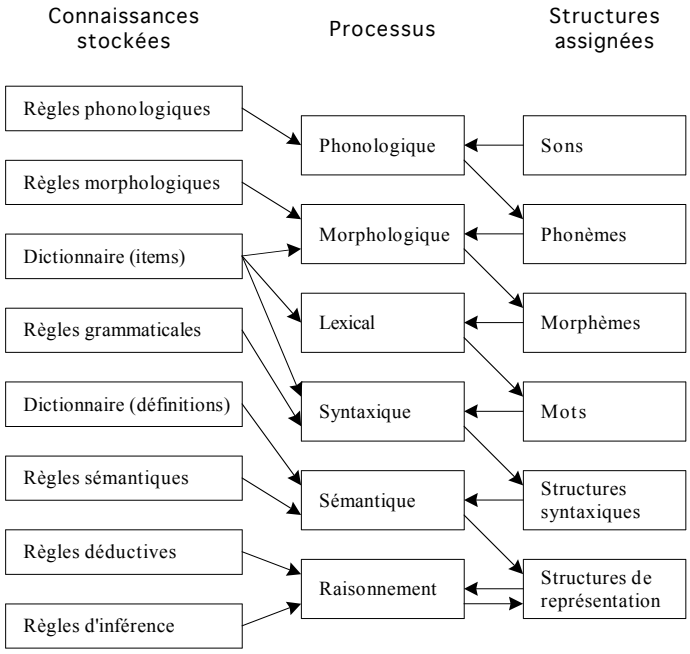


Fig. 1 – Modèle de compréhension linguistique

On constate à la lecture de celui-ci, que le niveau syntaxique constitue donc dans le domaine du TAL, avec l'analyse morphologique – qui réalise en fait à la fois l'analyse morphologique et l'analyse lexicale –, la première étape de l'analyse automatique d'une langue écrite .

Bien que la décomposition en tâches représente un grand principe de l'implantation informatique, certains chercheurs en TAL se sont rendu compte relativement tôt que les différents niveaux d'analyse ne pouvaient pas être totalement indépendants les uns des autres. Par exemple, des erreurs même très simples pour un francophone comme « *des livre* » ou « *un table* » ne peuvent pas être détectées par une analyse lexicale pure. Il va de soi que l'analyse de phrases telles que « *je les ai achetés / achetées* », qui exige de

trouver l'antécédent des pronoms pour les phénomènes d'accord, ou « *les questions de cette maladie qui me préoccupe / qui me préoccupent* », qui nécessite en plus de trouver le bon rattachement des groupes prépositionnels, ne se limitent pas au niveau syntaxique.

De plus, dans le cas du traitement automatique du japonais, cette première étape d'analyse a pour but la reconnaissance des unités morphologiques et/ou lexicales. Cette opération de reconnaissance des unités consiste sur le plan pratique, pour les langues n'ayant pas de séparateur comme le japonais, en une segmentation des phrases en unités morphologiques et/ou lexicales. Ces unités de base n'étant pas définies a priori comme elles peuvent l'être pour les langues dans lesquelles les phrases sont naturellement segmentées par des séparateurs – notamment l'espace –, la segmentation est une phase indispensable de presque tous les types de traitement d'analyse du japonais.

Notre étude sera constituée de cinq grande parties.

La première partie sera consacrée à l'étude de certaines notions linguistiques japonaises. En effet, il est bien évidemment indispensable, pour parler d'analyse automatique des langues, d'éclaircir les notions linguistiques, qui sont souvent spécifiques à une langue.

Du fait du statut extrêmement important de la segmentation dans l'ensemble des techniques d'analyse du japonais, nous dresserons dans la deuxième partie un panorama des méthodes de segmentation utilisées en japonais, pour comprendre comment les unités morphologiques ou lexicales, représentant les données pour l'analyse syntaxique, sont acquises.

Après cette introduction aux méthodes de segmentation, nous nous intéresserons aux connaissances de base de l'analyse syntaxique et à quelques algorithmes concrets.

Le noyau de la présente étude qui suit ces phases préparatoires, sera constituée de l'étude des analyseurs syntaxiques existants au Japon.

Nous étudierons précisément trois analyseurs syntaxiques automatiques japonais : le système SAX, l'analyseur MSLR et le système KNP. Le système d'analyse syntaxique SAX, développé dans un laboratoire du *Nara Institute of Science and Technology* (NAIST), est un analyseur générant un programme d'analyse syntaxique écrit en langage PROLOG. L'analyseur morpho-syntaxique MSLR, développé à l'Institut Technologique de Tokyo, est basé sur l'algorithme d'analyse syntaxique LR. Quant au système d'analyse syntaxique du japonais KNP de l'Université de Kyoto, il est caractérisé surtout par sa capacité de détection des structures (des syntagmes ou des propositions) de coordination.

Cette étude des analyseurs sera complétée par l'analyse des résultats obtenus par chaque système. Nous essayerons de saisir la grande ligne des problématiques de l'analyse syntaxique du japonais d'après cette étude.

Chapitre 1

Préliminaire : Notions de linguistique japonaise

1.1 Introduction

Comme beaucoup d'autres nations, les Japonais commencèrent très tôt à s'intéresser à leur langue, entraînant ainsi la création d'un grand nombre d'ouvrages parlant de cet outil de communication et moyen d'expression artistique.

Cet intérêt pour leur langue naquit dès l'Antiquité et le Moyen Âge, mais il ne se concrétisa pas tout de suite par des recherches linguistiques au sens strict du terme. Il s'agissait plutôt de recherches pratiques, réalisées non pas pour la recherche elle-même, mais dans d'autres buts, l'interprétation des ouvrages anciens et l'étude des poésies en étant les principaux. Au fur et à mesure que les recherches sur l'ancienne langue avançaient, les Japonais s'intéressaient également à la langue contemporaine, par comparaison avec celle-ci, constituant progressivement un ensemble d'études linguistiques. Ainsi à l'époque d'Edô (1600-1868), les premières théories générales de linguistique japonaise furent établies par des chercheurs comme Norinaga MOTOORI ou Nariakira FUJITANI.

Avec l'ouverture de l'archipel à la fin de l'époque d'Édo, les Japonais introduisirent abondamment les savoirs occidentaux, la linguistique n'y faisant pas exception. On vit alors, dès l'époque de Meiji (1868-1912), des travaux systématiques sur la grammaire japonaise contemporaine, appelée 口語文法 (*kôgo-bunpô*), sur la base de connaissances linguistiques occidentales et sur les travaux traditionnels. La grammaire que les Japonais apprennent aujourd'hui à l'école, dite 学校文法 (*gakkô-bunpô*, grammaire scolaire), est basée sur une des premières théories de ce type, établie par Shinkichi HASHIMOTO [20].

Ainsi, dans les termes linguistiques japonais cohabitent des notions familières pour les occidentaux et des concepts assez particuliers propres au ja-

ponais. Nous allons étudier dans ce chapitre différentes notions linguistiques japonaises que nous rencontrons dans les travaux sur le Traitement Automatique des Langues. Dans la mesure où les définitions de certaines notions varient très largement d'une théorie à l'autre, nous présenterons et utiliserons les définitions de la grammaire scolaire qui représente pour les Japonais les connaissances de base. Toutefois, cette grammaire scolaire étant également l'objet de nombreuses critiques, nous introduirons parfois d'autres points de vue, ceci afin de mieux comprendre les grandes lignes des problématiques de la linguistique japonaise.

Avant de commencer l'étude de la grammaire japonaise, nous allons faire une rapide présentation des systèmes d'écriture japonais.

1.2 Systèmes d'écriture japonais

1.2.1 Sens de l'écriture

Autrefois, le japonais s'écrivait verticalement de haut en bas et de droite à gauche. Lorsqu'une écriture horizontale, par exemple pour les noms de société ou de produit, était utilisée, il s'agissait également d'une écriture de droite à gauche. Après la seconde guerre mondiale, une réforme définit l'écriture des textes officiels comme horizontale et de gauche à droite. Depuis cette réforme, cette écriture horizontale de gauche à droite est appliquée à tous les textes officiels et généralisée à l'ensemble des textes. Aujourd'hui, l'écriture verticale est encore utilisée dans certains domaines tels que les journaux, les revues, les œuvres littéraires ou les manuels scolaires du japonais.

1.2.2 *kanji* et *kana*

Le japonais possède trois systèmes d'écriture différents : les caractères chinois, appelés 漢字 (*kanji*) et deux types de systèmes proprement japonais, dits 仮名 (*kana*), 平仮名 (*hiragana*) et 片仮名 (*katakana*).

Ces trois systèmes ont chacun un usage qui leur est propre. En effet, l'utilisation des *katakana* est limitée à l'écriture des mots étrangers (sauf dans le cas où ils jouent une fonction typographique). Les autres mots ayant un sens tels que les mots « pleins » et les radicaux, sont souvent écrits en idéogrammes, *kanji*, tandis que la partie variable des mots variants – comme les verbes ou les qualificatifs – et les mots n'ayant qu'une fonction grammaticale dits mots « vides » – tels que les particules –, ont comme particularité d'être représentés en caractères *hiragana*.

Ainsi, habituellement, un texte japonais est constitué de phrases où ces trois systèmes sont mélangés, appelées 漢字仮名交じり文 (*kanji-kana majiri bun*, phrase en écritures mélangées *kanji* et *kana*).

1.2.3 *rômaji* et symboles

Les Japonais utilisent également deux autres types de caractères : l'alphabet latin et différents symboles ([53]).

L'alphabet latin est appelé ローマ字 (*rômaji*).

On distingue les symboles utilisés selon deux types : les symboles de ponctuation (句切り符号, *kugiri-fugô*) et les symboles marquant la répétition (繰り返し符号, *kurikaeshi-fugô*).

Symboles de ponctuation 句切り符号 (*kugiri-fugô*)

1. 。 : *kuten* ou *maru*, point final ;
2. 、 : *tôten* ou *ten*, virgule ;
3. ， : *tôten* ou *kanma*, virgule (en écriture horizontale) ;
4. ・ : *heiretsuten* ou *nakaten*, point de coordination ou point centré ;
5. （） : *kakko*, parenthèses ;
6. 「」 : *kagi-kakko*, guillemets ;
7. 『』 : *futae-kagi*, doubles guillemets.

D'autres symboles tels que ―, ..., ～, → ou ｛｝ sont également utilisés.

Symboles de répétition 繰り返し符号 (*kurikaeshi-fugô*)

Ces symboles sont utilisés pour éviter d'écrire deux fois le même caractère. Les symboles de répétition sont 々, ゝ, ゞ et 〃.

Les symboles ゝ et ゞ, utilisés lors de la répétition de *hiragana*, ne peuvent être utilisés que dans le cas de l'écriture verticale de haut en bas et ce seulement dans le cas où la répétition se trouve à l'intérieur d'un même mot. Leur utilisation n'est plus très fréquente aujourd'hui.

En revanche, le premier symbole 々 peut être utilisé aussi bien dans l'écriture verticale que dans celle horizontale et son utilisation est assez courante. Il remplace la répétition de caractère *kanji*, comme par exemple 国々 (*kuniguni*, pays (pluriel)), 人々 (*hitobito*, gens (pluriel)), 年々 (*nen'nen*, chaque année), 日々 (*hibi*, chaque jour).

Le symbole 〃 peut également être utilisé dans les écritures verticales et horizontales, mais son utilisation est limitée aux tableaux : il signifie que le contenu de la case est équivalent à celui de la ligne précédente.

1.3 Unités linguistiques

La grammaire scolaire définit comme unités linguistiques : 文章 (*bunshô*, texte), 段落 (*danraku*, paragraphe), 文 (*bun*, phrase), 文節 (*bunsetsu*, syntagme) et 単語 ou 語 (*tango ; go*, mot).

Parmi ces cinq types d'unités, nous allons étudier plus précisément les trois derniers, car concernant le plus l'analyse syntaxique. En effet, un analyseur syntaxique reçoit comme données des mots, *tango* et les regroupe de manière à obtenir des syntagmes, *bunsetsu*. L'analyse syntaxique consiste à mettre en évidence les relations entre ces syntagmes constituant une phrase, 文 (*bun*).

Par ailleurs, les linguistes utilisent également le terme 形態素 (*keitaiso*, morphème), plus petite unité munie de sens.

1.4 *bun*

Les 文 (*bun*, phrase) sont définies, sur le plan formel, comme une séquence de caractères séparés par un point final, *kuten* (。).

1.4.1 Types de phrase

La grammaire scolaire définit trois types de phrases :

1. 単文 (*tanbun*, phrase simple) : elles ne contiennent pas de proposition ;
2. 複文 (*fukubun*, phrase composée) : elles contiennent une proposition subordonnée (従属節, *jôzokusetsu*)[1] ;
3. 重文 (*jûbun*, phrase double) : elles contiennent une proposition opposée (対立節, *tairitsusetsu*).

1.4.2 Éléments de phrase selon la grammaire scolaire

La grammaire scolaire distingue cinq types d'élément de phrase, dits 文の成分 (*bun no seibun*) selon leur fonction syntaxique.

1. 主語 (*shugo*, sujet) : élément constituant une phrase ou une proposition avec un prédicat ;
2. 述語 (*jutsugo*, prédicat) : élément constituant une phrase ou une proposition avec un sujet ;
3. 修飾語 (*shûshokugo*, mot qualifiant) : élément qualifiant d'autres éléments. On distingue deux types de *shûshokugo* :
   (a) 連用修飾語 (*renyôshûshokugo*, mot qualifiant un *yôgen*) : élément qualifiant un verbe ou un qualificatif, dit *yôgen* ;
   (b) 連体修飾語 (*rentaishûshokugo*, mot qualifiant un *taigen*) : élément qualifiant un substantif, dit *taigen* ;

[1] HASHIMOTO définit les propositions (節, *setsu*) comme des unités équivalent à une phrase, qui peuvent être un élément constituant d'une phrase. Il distingue deux grands types de proposition : les propositions subordonnées et les propositions opposées. Avec le premier type de proposition, la position de cette proposition et celle de la proposition principale ne peuvent pas être échangées, alors que pour le second type, l'échange de place est possible sans modification du sens de la phrase.

4. 接続語 (*setsuzokugo*, mot de conjonction) ;
5. 独立語 (*dokuritsugo*, mot indépendant).

1.4.3 Catégorisation d'éléments qualifiant le prédicat

La catégorisation d'éléments de phrase décrite précédemment est basée sur la théorie de HASHIMOTO et distingue les éléments qualifiant les *yôgen* en deux types : *shugo*, sujet, et *renyôshûshokugo*, mot qualifiant les *yôgen*.

Certains linguistes, comme Takao YAMADA, avancent une théorie qui distingue encore les éléments du deuxième type en deux classes, complément (補語, *hogo*) d'une part et mot qualifiant (修飾語, *shûshokugo*) d'autre part.

D'autres linguistes comme Minoru WATANABE, adoptent une position qui regroupe les deux classes définies par HASHIMOTO en une seule classe, dite 連用成分 (*renyô-seibun*, élément qualifiant un *yôgen*).

Les linguistes tels que Akira HORISHIGE ou Yasuo KITAHARA, appellent 補充語 (*hojû-go*, mot complément) ou 補充成分 (*hojû-seibun*, élément complément) les éléments constitués de la combinaison « substantif + particule de cas[2] » et 連用修飾語 (*renyôshûshokugo*, mot qualifiant un *yôgen*) ou 連用修飾成分 (*renyôshûshokuseibun*, élément qualifiant un *yôgen*) les éléments constitués d'un qualificatif à la forme converbale[3] ou d'un adverbe. Les 補充成分 (*hojû-seibun*, élément complément) se distinguent encore selon des critères formels ou des critères sémantiques. Avec les critères formels, un élément constitué d'un substantif et de la particule *ga* est appelé ガ格 (*ga-kaku*, cas de *ga*), un avec la particule *wo* est appelé ヲ格 (*wo-kaku*, cas de *wo*), et ainsi de suite. Avec les critères sémantiques, un élément constitué de la particule *ga* est appelé 主格 (*shu-kaku*, cas principal) si son référent est l'acteur, mais un élément avec la particule *no* peut également être *shu-kaku* dans un contexte enchâssé où la particule *no* remplace *ga*. Non seulement les types définis selon ces critères sémantiques varient énormément, mais en plus la catégorisation diffère d'un linguiste à l'autre.

Par ailleurs, il est important de noter que beaucoup de linguistes japonais contestent les théories s'appuyant sur l'opposition entre le sujet et le prédicat comme celle de HASHIMOTO, en affirmant que la phrase japonaise est basée non pas sur cette opposition – reflétant la grande influence des grammaires traditionnelles occidentales – mais sur l'opposition entre 主題 (*shudai*, thème) et le prédicat. En effet, le japonais est une langue possédant non seulement l'indication syntaxique du sujet, mais aussi l'indication syntaxique du thème. Cette caractéristique de double structure fut également remarquée par les linguistes occidentaux tels que Charles N. Li et Sandra A. Thompson dans [36]. Dans cet article, ils ont défini quatre types de langues selon la stratégie de construction des phrases, qui accorde de l'importance à

[2] Il s'agit d'une sous-catégorie de particules, qui regroupe les particules indiquant la fonction syntaxique du syntagme qui les précède.

[3] Il s'agit de la forme *renyô*, forme précédant un verbe (cf. section 1.7.1 page 21)

la notion de thème ou de sujet. Le japonais est classé avec cette typologie dans la catégorie des langues ayant aussi bien le caractère de prédominance du sujet que celui de prédominance thématique.

Les notions de sujet et de thème sont, encore aujourd'hui, souvent confondues tout en étant très différentes. La distinction entre ces deux notions est très importante pour les travaux sur la syntaxe dans le domaine du TAL, si bien que nous en reparlerons plus précisément à la fin de ce chapitre.

1.4.4 Notions de relation syntaxique : *kakari-uke*

係り受け

Les Japonais expriment traditionnellement la relation entre les éléments d'une phrase par les termes 係る (*kakaru*, se lier à ; s'accrocher à) et 受ける・承ける (*ukeru*, recevoir).

係る (*kakaru*, se lier à ; s'accrocher à) est défini dans [12] comme « le fait que l'effet d'un élément donné atteint un autre élément situé à une position postérieure ». En d'autres termes, il s'agit d'un phénomène dans lequel le sens d'un mot ou d'un syntagme détermine le sens d'un autre situé dans la partie postérieure de la phrase.

係り (*kakari*) est la nominalisation du terme *kakaru* et 受け (*uke*) celle de *ukeru*. 係り受け (*kakari-uke*) désigne donc la relation qualifiant-qualifié entre deux éléments.

Ces termes sont aujourd'hui très courants dans le domaine du TAL au Japon. De nombreux travaux sur la syntaxe ont été réalisés sur la base de la systématisation de ces relations. Cette relation de *kakari-uke* est généralement traduite par « relation de dépendance ». La notion de dépendance semblant bien correspondre au concept de *kakari-uke*, nous garderons cette traduction tout au long de la présente étude, et nous traduirons 受け (*uke*) par réception.

Les plus petits constituants directs d'une phrase sont appelés *bunsetsu*. Étudions maintenant cette unité, syntagme *bunsetsu*.

1.5 *bunsetsu*

La notion de 文節 (*bunsetsu*) provient de la théorie de HASHIMOTO [20]. Il définit cette unité comme la première unité que l'on obtient en segmentant une phrase et qui peut être un constituant de phrase. Il dit également que ce sont les plus petits éléments obtenus en segmentant au maximum une phrase, tout en conservant le statut d'énoncé de ces éléments. Les *bunsetsu* sont caractérisés, sur le plan formel, par la présence de coupures de syllabes immédiatement avant et après eux.

Cette unité ayant un support matériel phonétique pour justifier son statut, il est plus difficile de garder ce statut sur le plan sémantique. Il s'agit

d'un des points les plus critiqués de la théorie de HASHIMOTO. En effet, dans la phrase

桜　の | 花　は | もう | 散った。
(*sakura - no* | *hana - wa* | *mô* | *chitta*)
(cerisier - de | fleur - [thème] | déjà | tomber (passé))
(Les cerisiers ont déjà perdu leurs fleurs.)

le syntagme, *sakura-no*, ne s'attache pas au syntagme, *hana-wa*, mais il qualifie uniquement le substantif, *hana*. Ces syntagmes doivent donc être analysés comme :

(((*sakura no*) *hana*) *wa*)

pour que le syntagme, *sakura-no*, qualifie uniquement le substantif, *hana* et que la particule *wa* prenne cet ensemble pour l'attacher ensuite au prédicat de la phrase.

1.6 *tango*

HASHIMOTO dit qu'un syntagme peut être lui-même segmenté en unités linguistiques munies de sens, qu'il appelle 語 (*go*). Le centre du problème pour la définition de *go* ou 単語 (*tango*), se trouve dans la définition du statut des particules et des *jodôshi* (traduites ci-après par auxiliaires), unités qui, placées après un verbe ou un qualificatif, lui ajoutent certaines modalités. Certains considèrent les *jodôshi* et même, bien que plus rarement, les particules comme des unités n'appartenant pas aux *go*. Néanmoins, HASHIMOTO inclut ces deux catégories dans les *go*, en les distinguant des suffixes et des préfixes. Selon lui, les particules appartiennent aux *go*, car il ne considère comme des *setsuji* (接辞, unités n'appartenant pas aux *go*), que les unités qui interviennent dans la dérivation des *go*.

Nous pouvons définir deux types de classification des *tango*. La première classification s'appuie sur leur structure. On distingue alors trois catégories de *tango* selon ce critère. La seconde adopte comme critère leur fonction sur le plan grammatical. Les *tango* sont alors classés dans dix catégories lexicales, appelées 品詞 (*hinshi*). Nous allons maintenant étudier plus en détail ces deux modes de catégorisation des *go*.

1.6.1 Catégorisation selon leur structure

Les *tango* peuvent être catégorisés selon leur structure. On distingue alors les trois catégories suivantes.

1. 単純語 (*tanjungo*, mot simple) : mot constitué d'un seul morphème ;
2. 合成語 (*gôseigo*, mot synthétisé) : mot constitué de plusieurs morphèmes, se classant en :

(a) 複合語 (*fukugôgo*, mot composé) : mot constitué de plusieurs mots simples ;

(b) 派生語 (*haseigo*, mot dérivé) : mot constitué d'un mot et d'un *setsuji*.

1.6.2 Catégorisation selon leur fonction grammaticale

On classe dans la grammaire scolaire les *go* en dix catégories grammaticales. Les mots sont d'abord divisés en deux grandes classes : 自立語 (*jiritsugo*, mot autonome) et 付属語 (*fuzokugo*, mot annexe).

Comme nous l'avons mentionné dans la section 1.6 page précédente, il existe également un ensemble d'unités n'appartenant pas à ces deux catégories de *tango* et qui interviennent dans la dérivation des *tango* : les 接辞 (*setsuji*, préfixe et suffixe).

Première division : *jiritsugo* et *fuzokugo*

Cette distinction est basée sur le concept très ancien d'opposition des deux catégories désignées traditionnellement sous le nom de *shi* et *ji*.

shi* ou *kotoba* - *ji 詞・辞
Il s'agit de deux grandes catégories de la grammaire japonaise qui divisent les mots en deux ensembles selon leur nature grammaticale.

Le concept du *shi* existait déjà au Moyen Âge. Les *shi* sont des mots qui expriment conceptuellement et objectivement une idée. En revanche, les *ji* ne peuvent pas exprimer une idée, mais ajoutent au contenu exprimé par les *shi* la pensée subjective du locuteur, ou indiquent le rapport entre les mots. Akira SUZUKI, linguiste de l'époque moderne, définit les *shi* comme les mots désignant quelque chose, par opposition aux *ji*, traditionnellement appelés *tenioha*, qui ne désignent rien. De façon simplifiée, les *joshi* et les *jodôshi* appartiennent à la catégorie des *ji*, les autres à la catégorie des *shi*. Mais les différents éléments appartenant à chacune de ces catégories diffèrent selon les chercheurs.

La distinction de HASHIMOTO est plus formelle que la définition traditionnelle où il s'agit juste d'une catégorisation selon la nature du mot, objective ou subjective. La distinction entre *jiritsugo* et *fuzokugo* d'aujourd'hui est un héritage de l'opposition entre *shi* et *ji* de la théorie de HASHIMOTO.

Les *jiritsugo* (*shi*) sont définis comme des mots pouvant constituer à eux seuls un syntagme *bunsetsu*. Nous allons étudier par la suite les sous-catégories des *jiritsugo*.

Les *fuzokugo* (*ji*) sont définis selon HASHIMOTO comme des mots qui ne sont pas autonomes et qui sont toujours utilisés avec des mots qui doivent être autonomes.

Les *fuzokugo* sont divisés en deux catégories : ceux qui sont variables et ceux qui sont invariables. Les premiers désignent l'ensemble des 助動詞 (*jodôshi*), auxiliaires, et les seconds sont des particules, dits 助詞 (*joshi*).

Sous-catégories de *jiritsugo*

On distingue d'abord deux types de *jiritsugo*, variables et invariables.

Les *jiritsugo* variables sont traditionnellement appelés 用言 (*yôgen*). Les *yôgen* sont caractérisés par le fait qu'ils sont susceptibles d'être un prédicat. Cette notion de *yôgen* s'oppose à celle de 体言 (*taigen*), qui désigne les unités susceptibles d'être le sujet et qui correspond à la catégorie substantif.
On distingue dans la grammaire scolaire trois catégories de *yôgen* :

- unité exprimant l'action, l'effet ou l'existence :
  1. unité dont la forme de base se termine par *-u* : 動詞 (*dôshi*, verbe) ;
- unité exprimant la nature ou l'état :
  2. unité dont la forme de base se termine par *-i* : 形容詞 (*keiyôshi*, qualificatif en *i*) ;
  3. unité dont la forme de base se termine par *-da* ou *-desu* : 形容動詞 (*keiyôdôshi*, qualificatif en *na*).

Les *jiritsugo* invariables sont classés dans la grammaire scolaire en cinq catégories :

- unité pouvant être le sujet, 体言 (*taigen*) :
  1. unité désignant un objet ou un évènement : 名詞 (*meishi*, substantif) ;
- unité pouvant qualifier une autre unité :
  2. unité qui qualifie les *yôgen* : 副詞 (*fukushi*, adverbe) ;
  3. unité qui qualifie les *taigen* : 連体詞 (*rentaishi*, qualificatif invariable) ;
- unité pouvant être une conjonction :
  4. 接続詞 (*setsuzokushi*, conjonction) ;
- unité pouvant être un énoncé indépendant :
  5. 感動詞 (*kandôshi*, interjection) ;

1.6.3 Comparaison avec d'autres grammaires

Nous étudions ici également deux autres grammaires. La première est celle de Nagara [14]. Cette grammaire, tenant compte surtout des besoins des apprenants étrangers, présente une catégorisation assez différente du classement traditionnel. La seconde est la grammaire de Masuoka et Takubo [38]. Les deux analyseurs morphologiques du japonais les plus utilisés, JUMAN et CHASEN, reposent sur cette grammaire. Nous allons présenter les catégorisations proposées par ces deux grammaires, ainsi que l'adaptation de cette seconde à l'utilisation dans le domaine du TAL réalisée par JUMAN.

Catégorisation de la grammaire de NAGARA

L'ouvrage [14] présente une autre façon de catégoriser les *tango*, considérant cette nouvelle catégorisation comme reflétant la tendance générale d'aujourd'hui. L'auteur ajoute aux définitions de chaque catégorie des explications sur les différences par rapport à la grammaire scolaire et la raison de ces différences.

1. *jiritsugo*
   (a) 名詞 (*meishi*, substantif) : mots invariables désignant un objet ou un évènement. Ils peuvent être suivis de particules. Ils peuvent généralement être sujet. Les pronoms se comportant comme des substantifs sont considérés comme une sous-catégorie de *meishi*, exceptés les pronoms démonstratifs qui possèdent leur propre catégorie, *shijishi*.
   (b) 数詞 (*sûshi*, numéraux) : leur emploi principal est l'emploi adverbial[4]. Ne pouvant pas recevoir de mots qualifiant les *taigen*, ils constituent une catégorie indépendante.
   (c) イ形容詞 (*i-keiyôshi*, qualificatif en *i*) : cette catégorie correspond à la catégorie *keiyôshi* de la grammaire scolaire. Ce sont des mots variables, qui expriment un sentiment, une sensation, un état d'objet ou d'événement, pouvant constituer à lui seul le prédicat. Ces propriétés sont communes aux qualificatifs en *i* et en *na* – mais ils diffèrent par le mode de changement de forme. La forme dite finale se termine par *i*.
   (d) ナ形容詞 (*na-keiyôshi*, qualificatif en *na*) : cette catégorie correspond à la catégorie *keiyôdôshi* de la grammaire scolaire. Étant donné que leurs radicaux ont un très fort caractère d'indépendance et que leurs variations ressemblent à celles de la construction « substantif + *da* (copule) », ils sont parfois considérés comme une sous-catégorie de substantif, alors désignés sous le nom de « substantifs en *na/ni* » ou substantifs qualificatifs.

[4] Les numéraux sont généralement considérés comme des substantifs.

(e) 動詞(*dôshi*, verbe) : mots variables pouvant constituer à eux seuls le prédicat. La forme dite finale se termine par une syllabe dont la voyelle finale est -u. Ils expriment, à la différence des qualificatifs, l'état d'un événement en le situant temporellement.

(f) 副詞 (*fukushi*, adverbe) : *jiritsugo* jouant principalement le rôle de qualification des *yôgen*.

(g) 指示詞 (*shijishi*, démonstratif) : mots constituant le système dit こそあど (*ko-so-a-do*[5]). En général, ils se répartissent dans les catégories pronom, qualificatif invariable et adverbe, mais on considère ici qu'ils appartiennent à la même catégorie et qu'ils jouent chacun dans la phrase une fonction différente de complément, de mot qualifiant un *taigen* ou de mot qualifiant un *yôgen*.

(h) 接続詞 (*setsuzokushi*, conjonction) : *tango* n'ayant pas de fonction de qualification, et reliant les *tango* et les phrases.

(i) 間投詞 (*kantôshi*, interjection) : peuvent constituer à eux seuls une phrase. En tant qu'élément de phrase, ce sont des éléments indépendants. Ils n'ont ni variation de forme, ni fonction de qualification, ni fonction de connexion.

2. *fuzokugo*

(a) 助詞 (*joshi*, particule) : *fuzokugo* invariables. Ils suivent toujours un autre *tango*. Ils expriment non pas un objet ni un état, mais la relation entre l'objet et l'évènement établie par le locuteur.

(b) 助動詞 (*jodôshi*, auxiliaire) : *fuzokugo* variables. Ils suivent toujours un autre *tango*. Ils sont utilisés pour préciser la façon de juger du locuteur.

(c) コピュラ (*kopyura*, copule) : à la suite d'un substantif, ils constituent le prédicat. On les appelle également 判定詞 (*hanteishi*). Dans la grammaire scolaire, ils appartiennent en général à une sous-catégorie de *jodôshi*, dite 断定の助動詞 (*dantei no jodôshi*), *jodôshi* d'affirmation.

Catégorisation de la grammaire de Masuoka et Takubo

La grammaire de Masuoka et Takubo [38] distingue onze catégories lexicales qu'ils ont définies selon leur fonction dans la phrase comme suit :

1. 動詞(*dôshi*, verbe) : peut constituer à lui seul le prédicat.
2. 形容詞 (*keiyôshi*, qualificatif) : peut constituer à lui seul le prédicat et peut qualifier un syntagme nominal.

[5]Il s'agit de morphèmes exprimant la position physique ou psychologique. En s'associant avec d'autres sons ou des caractères, ils constituent les pronoms démonstratifs, les qualificatifs démonstratifs et les adverbes démonstratifs.

3. 判定詞 (*hanteishi*, mot de jugement) : à la suite d'un substantif, constitue le prédicat.
4. 助動詞 (*jodôshi*, auxiliaire) : à la suite d'un prédicat, constitue le prédicat complexe.
5. 名詞 (*meishi*, substantif) : peut être l'élément principal du thème ou du complément.
6. 副詞 (*fukushi*, adverbe) : peut qualifier le syntagme verbal.
7. 助詞 (*joshi*, particule) : à la suite d'un substantif, constitue le thème ou un complément, ou relie des substantifs ou des syntagmes.
8. 連体詞 (*rentaishi*, qualificatif de substantif) : qualifie le syntagme nominal.
9. 接続詞 (*setsuzokushi*, conjonction) : relie les phrases.
10. 感動詞 (*kandôshi*, mot d'émotion) : peut constituer à lui seul une phrase.
11. 指示詞 (*shijishi*, démonstratif) : désigne les personnes ou les choses de manière déictique ou anaphorique.

Catégorisation de JUMAN

JUMAN définit deux types de catégories. Le premier type est appelé 形態品詞 (*keitai-hinshi*, catégorie morphologique) et le second type, sous-catégorie de ce premier, 形態細品詞細分類(*keitai-hinshi-saibunrui*, sous-catégorie morphologique). Chaque ensemble appartenant à une des catégories de ces deux niveaux est appelé *hinshi* (catégorie).

JUMAN distingue quatorze catégories, *keitai-hinshi*, les onze premières étant basées sur les catégories lexicales de la grammaire de Masuoka et Takubo [38][6], et les trois dernières catégories étant définies comme :

1. 特殊(*tokushu*, spécial) : comprend les symboles de ponctuation, les autres symboles et les guillemets.
2. 接頭辞 (*settôji*, préfixe) : précéde certains mots et constitue avec lui un autre mot sans modifier la catégorie du mot auquel il est rajouté.
3. 接尾辞 (*setsubiji*, suffixe) : suit certains mots pour constituer un autre mot et modifier la catégorie du mot en une catégorie donnée.

Du fait de besoins bien définis, dus aux contraintes informatiques, JUMAN catégorise également les suffixes et les préfixes, que les linguistes ne considèrent pourtant pas comme des unités lexicales. Cependant, cette catégorisation est tout à fait cohérente dans la mesure où JUMAN appelle ces catégories, comme il a déjà été dit, catégories morphologiques. Toutefois, la

[6] À noter que dans le cas de JUMAN, les expressions se comportant comme des particules, telles que という (*toiu*, appelé), に対して (*nitaishite*, pour), だけでなく (*dakedenaku*, non seulement), sont considérées comme des particules.

notion même de « morphème » pour JUMAN n'est pas claire car la plupart des unités traitées, définies comme *keitaiso* (morphèmes), ne sont pas toujours des unités morphologiques. Le mot synthétisé « *o-cha* » ([embellissement] + thé) est traité comme un morphème et « *tabe-ta* » (v. manger (radical) + [passé ; accompli]) également. En fait, ces unités correspondent probablement aux entrées du dictionnaire utilisé.

1.7 Conjugaison : *katsuyô*

Une caractéristique essentielle du *yôgen* est la variation morphologique. Nous constatons dans plusieurs ouvrages dès le Moyen Âge l'intérêt des écrivains pour les changements de forme qui apparaissaient dans les terminaisons de mots, représentant des variations entre les sons d'une même colonne de la table 五十音図 (*gojûon-zu*, table des cinquante sons). Ce changement fut exprimé par le mot *hataraku*. Le terme *katsuyô* commença à être employé à partir de l'époque Edo par Norinaga MOTOORI.

Avant d'entrer dans l'étude des types et des formes de conjugaison, présentons la table *gojûon-zu*, fortement liée à la définition des termes concernant la conjugaison.

Table des cinquante sons : *gojûon-zu* 五十音図

わ	ら	や	ま	は	な	た	さ	か	あ
wa	ra	ya	ma	ha	na	ta	sa	ka	a
	り		み	ひ	な	ち	し	き	い
	ri		mi	hi	ni	chi	shi	ki	i
	る	ゆ	む	ふ	な	つ	す	く	う
	ru	yu	mu	fu	nu	tsu	su	ku	u
	れ		め	へ	な	て	せ	け	え
	re		me	he	ne	te	se	ke	e
を	ろ	よ	も	ほ	な	と	そ	こ	お
wo	ro	yo	mo	ho	no	to	so	ko	o

C'est un tableau dans lequel l'ensemble des caractères syllabiques japonais, *kana*, sont rangés en dix colonnes de cinq lignes.

On appelle 行 (*gyô*) les colonnes de cinq caractères et 段 (*dan*) les lignes de dix caractères. Les *gyô* réunissent les *kana* qui représentent une syllabe constituée d'un même son consonnantique, et dans une même *dan* on trouve les *kana* représentant une syllabe constituée d'une voyelle commune. Ainsi, par exemple, ア行 (*a-gyô*, colonne de *a*), la colonne située la plus à droite, désigne la colonne rassemblant les cinq syllabes あいうえお (*a-i-u-e-o*) et ア段 (*a-dan*, ligne de *a*), la première ligne en partant du haut, désigne la ligne rassemblant les syllabes あかさたなはまやらわ (*a-ka-sa-ta-na-ha-ma-ya-ra-wa*).

1.7.1 Formes de conjugaison : *katsuyô-kei*

Les différentes formes des *yôgen* et des *jodôshi*, variant selon le type de *go* qui les suit, sont appelées 活用形 (*katsuyô-kei*). La partie invariable est appelée 語幹 (*gokan*, radical) et la partie variable 活用語尾 (*katsuyô-gobi*, terminaison). Dans la grammaire scolaire, six formes sont définies selon les mots qui les suivent. Les exemples suivants sont des formes conjuguées du verbe 書く (*kaku*, écrire)[7] :

1. 未然形 (*mizen-kei*, forme *mizen*) : suivie par ない(*nai*, [négation]) – 書か・ない (*kaka-nai*), う／よう (*u/yô*, [volitif]) – 書こ・う (*kako-u*) ;
2. 連用形 (*renyô-kei*, forme précédant les *yôgen*) : suivie par た(*ta*, [passé ; accompli]) – 書い・た (*kai-ta*), ます(*masu*, [politesse]) – 書き・ます (*kaki-masu*) ;
3. 終止形 (*shûshi-kei*, forme finale) : suivie par la particule de connexion と(*to*, lorsque) – 書く・と (*kaku-to*) ;
4. 連体形 (*rentai-kei*, forme précédant les *taigen*) : suivie par こと(*koto*, ce qui) – 書く・こと (*kaku-koto*), とき(*toki*, quand) – 書く・とき (*kaku-toki*) ;
5. 仮定形 (*katei-kei*, forme de condition) : suivie par ば(*ba*, [condition]) – 書け・ば (*kake-ba*) ;
6. 命令形 (*meirei-kei*, forme impérative) – 書け (*kake*).

Certaines formes telles que les formes *mizen*, *kaka* et *kako*, ou une des formes *renyô*, *kai*, ne pouvant pas être utilisées toutes seules, doivent être considérées non pas comme des *tango* mais comme des morphèmes. Si bien que certaines grammaires considèrent comme forme conjuguée la partie comprenant également le *fuzokugo* qui les suit. L'ouvrage [14] réorganise les formes conjuguées de la grammaire scolaire, de manière à s'adapter particulièrement à l'enseignement du japonais comme langue étrangère, de la façon suivante :

1. ル形 (*ru-kei*, forme en *ru*), 辞書形 (*jisho-kei*, forme du dictionnaire) ou 基本形 (*kihon-kei*, forme basique) : forme finale et forme *rentai* de la grammaire scolaire ;
2. 連用形 (*renyô-kei*, forme précédant les *yôgen*) : forme *renyô* ;
3. タ形 (*ta-kei*, forme en *ta*) : forme *renyô* + *ta* ;
4. テ形 (*te-kei*, forme en *te*) : forme *renyô* + *te* ;
5. マス形 (*masu-kei*, forme en *masu*) : forme *renyô* + *masu* ;
6. 否定形 (*hitei-kei*, forme négative) : forme *mizen* + *nai* ;
7. 意志形 (*ishi-kei*, forme volitive) : forme *mizen* + *u/yô* ;

[7] Le point ・ marque la frontière entre le mot *kaku* et le *fuzokugo*, ou tout autre mot qui le suit.

8. 仮定形 (*katei-kei*, forme de condition) : forme de condition + *ba* ;
9. 命令形 (*meirei-kei*, forme impérative) : forme impérative.

JUMAN les définit encore plus précisément en s'appuyant comme toujours sur des critères totalement formels. Les verbes, les qualificatifs de type *i*, les auxiliaires et les suffixes de type verbal ont les formes conjuguées suivantes :

1. forme basique ;
2. forme *mizen* ;
3. forme volitive ;
4. forme impérative ;
5. forme de condition de type basique ;
6. forme *renyô* de type basique ;
7. forme en *ta* ;
8. forme de conjecture de type *ta* ;
9. forme de condition de type *ta* ;
10. forme *renyô* de type *ta* ;
11. forme *renyô* de type *tari* ;
12. forme *renyô* de type *cha*.

Les formes ci-dessus ne sont pas toujours toutes définies. Certaines catégories ont également des formes de type ancien japonais.

Les qualificatifs de type *na*, de type *nano*, de type *na* particulier et les mots de jugement se conjuguent sur la base des formes décrites précédemment, mais avec trois types de terminaisons possibles :

1. système *da* ;
2. système *dearu* ;
3. système *desu*.

1.7.2 Types de conjugaison

Les mots subissent une variation morphologique suivant un certain modèle donné. On appelle ce modèle 活用型 (*katsuyô-gata*, type de conjugaison).

Nous présentons les types de conjugaison définis dans la grammaire scolaire, ainsi que ceux définis dans le système JUMAN.

Types de conjugaison de la grammaire scolaire

Les verbes ont cinq types de conjugaison. Les qualificatifs en *i* et en *na* ont chacun un seul type de conjugaison et les *jodôshi* se conjuguent comme un des types de verbe ou de qualificatif, sauf quelques exceptions.

Les cinq types de conjugaison des verbes sont les suivants :

1. 五段活用 (*godan-katsuyô*, conjugaison de type 5 *dan*) : la variation de forme se réalise sur les cinq lignes de la table *gojûon* ;
2. 上一段活用 (*kami-ichidan-katsuyô*, conjugaison de type *kami* 1 *dan*) : la variation de forme se réalise sur la ligne de *i* ;

3. 下一段活用 (*shimo - ichidan - katsuyô*, conjugaison de type *shimo* 1 *dan*) : la variation de forme se réalise sur la ligne de *e* ;
4. カ行変格活用 (*kagyô-henkaku-katsuyô*, conjugaison de type *kahen*) : variation de forme exceptionnelle, particulière au verbe 来る (*kuru*, venir) ;
5. サ行変格活用 (*sagyô-henkaku-katsuyô*, conjugaison de type *sahen*) : variation de forme exceptionnelle, particulière au verbe する (*suru*, faire) ;

Les verbes qui obéissent à la variation de type 5 *dan* sont appelés 五段動詞 (*godan-dôshi*, verbes 5 *dan*) dans la grammaire scolaire et 子音動詞 (*shiin-dôshi*, verbes consonantiques) ou verbes du premier groupe, dans d'autres grammaires.

Les verbes qui obéissent à la variation de type 1 *dan* sont appelés 上一段動詞 (*kami-ichidan-dôshi*, verbes *kami* 1 *dan*) ou 下一段動詞 (*shimo-ichidan-dôshi*, verbes *shimo* 1 *dan*) dans la grammaire scolaire et 母音動詞 (*boin-dôshi*, verbes vocaliques) ou verbes du deuxième groupe, dans d'autres grammaires.

Les verbes qui obéissent à une variation de type exceptionnelle sont appelés カ行変格活用動詞 (*kagyô-henkaku-katsuyô-dôshi*, verbes *kahen*) ou サ行変格動詞 (*sagyô-henkaku-katsuyô-dôshi*, verbes *sahen*) dans la grammaire scolaire et verbes irréguliers ou verbes du troisième groupe, dans d'autres grammaires.

Types de conjugaison de JUMAN

On distingue 17 types de conjugaison pour les verbes, 6 pour les qualificatifs, 5 pour les auxiliaires, 3 pour les suffixes de type verbal et 1 pour les mots de jugement.
Pour les verbes :

1. type vocalique ;
2. type consonantique sur *ka* ;
3. type consonantique sur *ka* avec gémination ;
4. type consonantique sur *ga* ;
5. type consonantique sur *sa* ;
6. type consonantique sur *ta* ;
7. type consonantique sur *na* ;
8. type consonantique sur *ba* ;
9. type consonantique sur *ma* ;
10. type consonantique sur *ra* ;
11. type consonantique sur *ra* en *i* ;
12. type consonantique sur *wa* ;
13. type consonantique sur *wa* avec modification de son de l'ancien japonais ;
14. type consonantique sur *ka* particulier, appelé *kahen* ;
15. type consonantique sur *ka* particulier en *kanji* ;
16. type consonantique sur *sa* particulier, appelé *sahen* ;
17. type consonantique sur *za*.

Pour les qualificatifs :

1. type *i* sur *a, u, o* ;
2. type *i* sur *i* ;
3. type *i* sur *i* particulier ;
4. type *na* ;
5. type *nano* ;
6. type *na* particulier.

Pour les auxiliaires :

1. type *nu* ;
2. type *darô* ;
3. type *sôda* ;
4. type *ku* ;
5. type invariable.

Pour les suffixes de type verbal :

1. type *masu* ;
2. type *uru* ;
3. type *uru* en kanji ;

1.8 Notion de thème

La grammaire scolaire qui distingue dans la phrase deux éléments, le sujet – ce dont on parle – et le prédicat – ce qui l'explique –, ne peut pas expliquer la différence existant entre un élément suivi d'une particule *ga* et un autre introduit par une particule *wa*. Par exemple, elle n'a pas de moyen pour expliquer la différence entre les syntagmes en *ga*, ピエールが, et en *wa*, ピエールは des phrases suivantes :

ピエール | が | 来た。
(Pierre | (*ga*) | venir (passé))
« Pierre est venu »
ピエール | は | 来た。
(Pierre | (*wa*) | venir (passé))
« Pierre, il est venu »
ピエール | は | 呼んだ。
(Pierre | (*wa*) | inviter (passé))
« Pierre, (je) l'ai invité »

En réalité, ces deux particules sont des unités de niveau fort différent. La particule *ga* appartient à la sous-catégorie des particules appelée 格助詞 (*kaku-joshi*, particule de cas), qui regroupe les particules indiquant la fonction syntaxique du syntagme qui les précède. En revanche, la particule *wa* appartient à la sous-catégorie des particules appelée 副助詞 (*fuku-joshi*, particule secondaire) qui regroupe les particules ayant comme fonction d'ajouter un sens supplémentaire. La particule *wa* a comme fonction de transformer le syntagme qu'elle précède en thème à propos duquel on parle.

Les particules *fuku-joshi* peuvent suivre différents syntagmes, y compris les syntagmes postpositionnels terminés par une particule de cas. Si bien que l'on peut tout à fait avoir une séquence avec une particule de cas suivie d'une ou même deux *fuku-joshi*. Cependant, les particules *ga* et *wo*[8], lorsqu'elles sont suivies de la particule *wa*, disparaissent, ce qui provoque souvent une confusion quant aux niveaux des particules *kaku-joshi* et *fuku-joshi*, en fait très différents.

Du fait de cette situation de confusion, des linguistes japonais comme MIKAMI ont même avancé une théorie visant à abolir le terme « sujet ». Cette idée extrême ne signifie cependant pas qu'ils niaient l'existence, ou la prédominance par rapport à d'autres arguments, du sujet dans la phrase japonaise. Akira MIKAMI dit par exemple dans [42] :

> « Le syntagme "X *ga*" étant très important par rapport aux syntagmes "X *wo*" ou "X *ni*[9]", il est normal de lui accorder de l'importance et je ne suis pas contre cette attitude. [...] L'important est de lui accorder une importance **de manière appropriée**.
>
> "La phrase est constituée de deux parties, sujet et prédicat" est un concept particulier propre aux langues européennes et ne convient pas au japonais. Malgré cette réalité, la plupart des gens considèrent aveuglement, avec leur complexe d'un pays sous-developpé, cette opposition de sujet-prédicat comme quelque chose de noble ».

En effet, les variations morphologiques des verbes, adjectifs, copules ou autres *jodôshi*, dépendent, à la différence des langues européennes, non pas du sujet mais de l'élément qu'ils qualifient (forme de qualification d'un substantif, d'un prédicat ou forme finale s'ils ne qualifient rien, etc.). Dans une phrase en japonais, le sujet n'a donc pas de prédominance absolue par rapport aux autres éléments, mais seulement une prédominance relative justifiée par le fait que, par exemple, tous les verbes prennent un sujet ou que les expressions de politesse varient en fonction du sujet, etc.

MIKAMI souligna ainsi l'importance d'établir une grammaire japonaise basée sur la notion de thème. Il n'est cependant pas le premier linguiste à s'être rendu compte du statut tout à fait différent des particules *ga* et *wa*. On trouve déjà dans l'ouvrage [41] de MATSUSHITA, publié en 1928, une remarque sur cette différence.

Après cet aperçu succinct des différence de statut des particules *ga* et *wa*, nous allons maintenant nous intéresser à la fonction exacte du « thème ».

Yasuo KITAHARA qui affirme comme MIKAMI que « en japonais l'élément en relation avec le prédicat est non pas le sujet mais le thème », explique dans le chapitre 「文の構造」 de [45] (pages 31 - 82) ainsi la fonction du thème :

[8] Il s'agit d'une particule marquant le COD.

[9] Il s'agit d'une particule marquant le destination.

> « Le 主題 (thème) "X *wa*" entraîne l'achèvement de la construction d'une phrase par interaction mutuelle avec la fin de phrase située dans la partie de prédicat, tout en conservant (ou "représentant" selon MIKAMI) la fonction grammaticale "X *ga*", "X *wo*", "X *ni*" ou "X *de*" qu'il occuperait dans la phrase en ～コト (*koto*, le fait que)[10]. Étant donné que le thème entraîne par interaction mutuelle avec le prédicat l'accomplissement de la construction d'une phrase, il correspond au sujet dans les langues comme l'anglais ».
>
> Dans la phrase :
>
> この本 | は、 | 父 | が | 買ってくれました。
> (ce livre | (*wa*) | mon père | [sujet] | m'acheter (passé))
> « Ce livre, mon père me l'a acheté »
>
> on introduit une information nouvelle 父が買ってくれました (mon père me l'a acheté) à propos du thème (information connue) この本は (ce livre).

Cette définition de la fonction de *wa* présente deux problèmes. D'une part, KITAHARA utilise le concept 既知の情報 (information connue) sans le définir, et d'autre part, il existe beaucoup de phrases dans lesquelles la fonction grammaticale du syntagme suivie de *wa* dans le dictum[11] ne peut pas être définie.

夏 | は、 | 仕事 | が | 忙しい。
(été | (*wa*) | travail | (*ga*) | être chargé (adj.))
« En été, (j') ai beaucoup de travail »

Il y a également des phrases dans lesquelles la fonction du syntagme suivi de *wa* coïncide avec la fonction assurée par un autre syntagme.

新聞 | は、 | 読売 | を | 読んでいます。
(journal | (*wa*) | Yomiuri | [C.O.D.] | lire [habitude])
« Mon quotidien, c'est le Yomiuri »

Il est également difficile d'appliquer cette définition aux structures telles que :

田中さんの家 | は、 | 庭 | が | 広い。

[10] N.d.T. : afin de connaître la particule marquant la fonction du syntagme, omise du fait de l'utilisation de la particule *wa*, on transforme la phrase en une structure enchâssée, comme avec *koto*, structure dans laquelle la particule *wa* ne peut pas apparaître.

[11] Selon Minoru WATANABE, linguiste japonais, la phrase est constitué de 叙述 (*jojutsu*) et de 陳述 (*chinjutsu*). D'après lui, 叙述 (*jojutsu*) est le contenu d'un pensée ou d'un fait, et 陳述 (*chinjutsu*) une fonction syntaxique qui détermine, en prenant le contenu de 叙述 (*jojutsu*) comme matière, le rapport entre le contenu de 叙述 (*jojutsu*) et le sujet parlant. Nous traduisons dorénavant ces notions, pour faciliter la compréhension, respectivement par dictum et par modus.

(maison de M. TANAKA | (*wa*) | jardin | (*ga*) | grand (adj.))
« La maison de M. TANAKA a un grand jardin »
田中さん | は、 | 家族 | が | フランス | に | 住んでいる。
(M. TANAKA | (*wa*) | famille | (*ga*) | France | [lieu] | habiter)
« M. TANAKA, sa famille habite en France »

Par ailleurs, dans [31] KURODA présente la particule *wa* comme marque de sujet au sens logique et justifie la distinction du jugement thétique et du jugement catégorique faite par Franz BRETANO et Anton MARTY, en montrant la correspondance des phrases japonaises sans syntagme en *wa* avec le jugement de premier type et la correspondance des phrases japonaises avec syntagme en *wa* avec le jugement de deuxième type.

Il est difficile, voire même impossible, de définir exactement la fonction de thème de façon aussi restreinte. Nous adoptons donc pour le moment la définition classique du thème, insuffisante sans doute, à savoir « ce à propos de quoi il est question », en reportant l'étude plus précise de cette notion aux années à venir.

Chapitre 2

Méthodes de segmentation

2.1 Introduction

Nous consacrons ce chapitre aux méthodes de 単語分割 (*tango - bunkatsu*, mot - segmentation), segmentation en unités lexicales, qui, liées fortement à notre sujet principal, sont également une connaissance nécessaire préalable à l'étude de l'analyse syntaxique.

La segmentation est généralement comprise dans une étape d'analyse, appelée 形態素解析 (*keitaiso-kaiseki*, morphème[1] - analyse automatique). L'analyse morphologique est une phase essentielle de toute application d'analyse automatique des langues écrites. Cette première étape est souvent unifiée avec l'étape lexicale et plutôt désignée sous le nom d'analyse lexicale (terme qui n'est pas très courant dans le domaine du TAL au Japon). Elle a pour fonction de reconnaître les unités morphologiques, de lemmatiser les unités et d'identifier les catégories lexicales.

Dans le cas précis du traitement automatique du japonais, cette première étape d'analyse est caractérisée par le fait qu'elle a essentiellement pour but la reconnaissance des unités morphologiques et/ou lexicales, contrairement aux langues comme le français où l'analyse morphologique est utilisée pour catégoriser les unités.

En effet, dans le cas des langues possédant un certain nombre de séparateurs graphiques, notamment une espace, la segmentation d'une phrase en unités la composant – la phrase représentant l'unité de travail de l'étape suivante qu'est l'analyse syntaxique – pose moins de problème. Pour ces

[1] Le terme *keitaiso* est une notion linguistique définie comme la plus petite unité significative, comme l'est le terme morphème en français. Mais dans le domaine du TAL, l'analyse morphologique ne traite pas forcément que les morphèmes. En fait, le terme *keitaiso* est parfois utilisé sans respecter la définition, le rendant ainsi synonyme de *go* ou *tango*, termes japonais équivalents au « mot ». Cet usage illégitimement étendu est probablement lié au fait que les chercheurs japonais en TAL n'utilisent pas le terme « analyse lexicale ». Nous gardons tout de même la traduction « analyse morphologique » du terme *keitaiso-kaiseki*.

langues, les problèmes de reconnaissance des unités se localisent dans les mots graphiques (mots séparés par un symbole typographique) qui ne coïncident pas avec des unités linguistiques. Dans le cas de l'analyse du français, il s'agit non pas d'une segmentation, mais plutôt d'une reconstitution de mots « discontigus ».

En revanche, de par l'absence de tout séparateur entre les unités, le japonais nécessite tout d'abord une phase lourde de segmentation des phrases japonaises en écriture contiguë (べた書き, *betagaki*), dite traitement en écriture segmenté (分かち書き, *wakachigaki*).

Nous allons maintenant dresser un panorama de l'état de l'art des méthodes de segmentation au Japon. Nous nous intéresserons tout d'abord aux applications concrètes de segmentation. Nous étudierons ensuite les ambiguïtés du japonais lors de la segmentation des phrases en *tango*, avant de passer à la présentation des différentes méthodes utilisées au Japon pour la reconnaissance des unités lexicales permettant de segmenter les phrases japonaises.

Certains exemples présentés dans ce chapitre sont tirés de [39] et [50].

2.2 Applications de la segmentation en *tango*

On peut citer deux grands types d'application nécessitant des techniques de segmentation.

2.2.1 Système de saisie du japonais

La première application est le système de saisie du japonais. Un des trois systèmes d'écriture utilisés par les Japonais, les *kanji*, possédant deux mille caractères usuels – nombre qui ne couvre cependant pas la totalité des caractères utilisés dans les journaux –, il est impossible, du moins très difficile, de concevoir un système permettant aux non spécialistes, de saisir directement les caractères souhaités.

Différentes pistes furent ainsi suivies par les chercheurs – qui avaient ressenti très tôt la nécessité de pouvoir utiliser l'écriture japonaise sur ordinateur – pour concevoir un système permettant de saisir un texte en japonais sur ordinateur : tablette graphique, reconnaissance vocale, ou saisie directe sur un clavier traditionnel à l'aide de combinaisons de touches.

Les précurseurs développèrent finalement un système interactif de saisie utilisant un clavier constitué de touches représentant les *kana*, et qui adoptait la méthode de transformation des *kana* en *kanji*, appelée 漢字仮名変換 (*kanji kana henkan*) .

Aujourd'hui, les utilisateurs peuvent saisir les *kanas* directement avec un clavier japonais, ou phonétiquement avec un clavier latin.

Cette méthode de transformation en écriture standard des représentations phonétiques est également utilisée dans d'autres langues ayant beau-

coup de caractères comme le chinois ou le coréen. Les programmes dédiés à cette opération sont appelés *Input Method Editor* (IME ci-après).

Les opérations des programmes d'IME du japonais consistent à segmenter en *tango* la phrase en écriture *kana* ou en alphabet latin, appelé *rômaji*, pour choisir ensuite la représentation correspondante de chaque mot. Pour une représentation phonétique donnée, on obtient, selon la segmentation, différentes représentations en écriture « correcte » du japonais, dite 漢字仮名交じり文 (*kanji-kana majiri bun*, phrase en écritures mélangées de *kanji* et *kana*). Par exemple, la séquence :

ここではきものをぬぐ (*ko ko de ha ki mo no wo nu gu*)

peut être segmentée de deux façons différentes :

1. ここ | で | はきもの | を | ぬぐ
   (ici - [lieu] - chaussures - [COD] - enlever)
   → ここで履き物を脱ぐ (On se déchausse ici.)
2. ここ | で | は | きもの | を | ぬぐ
   (ici - [lieu] - [thème] - habits - [COD] - enlever)
   → ここでは着物を脱ぐ (Ici, on se déshabille.)

Ces deux segmentations étant toutes les deux tout à fait correctes, il est impossible de concevoir un système avec un taux d'erreurs de 0 pour cent. Il y a toujours des choix à confier à l'utilisateur.

2.2.2 Programme d'attribution de lecture correcte à un texte

Le second type d'application concerne la synthèse de la parole à partir d'un texte. Pour mener à bien cette opération, il faut tout d'abord obtenir la « lecture » correcte de la représentation écrite, ce qui est réalisé par un programme appelé 読み振りプログラム (*yomifuri puroguramu*, programme d'attribution de la lecture). Ce programme réalise d'abord la segmentation en *tango* de phrases en écritures mélangées, et il choisit ensuite la lecture de chaque mot. Pour cette opération également, la segmentation est une des sources d'ambiguïté sémantique.

Comme nous venons de le voir, la segmentation est le tout premier problème rencontré dans le traitement du japonais, et son domaine d'application est très large.

2.3 Ambiguïté de segmentation

Dans n'importe quel texte japonais, beaucoup de mots composés sont souvent constitués uniquement de caractères chinois, *kanji*. En effet, en japonais il est possible de créer assez librement des mots composés en associant plusieurs substantifs (éventuellement avec des préfixes et/ou des suffixes). Nous nous intéressons ici au problème de segmentation des phrases

japonaises en utilisant comme exemple la segmentation en *tanjungo*, mots simples, de ces mots composés.

La consultation du dictionnaire pour analyser le mot 全国都道府県議会議長会 donne les 19 unités concernées suivantes :

1. 全 (*zen*, tout)
2. 全国 (*zenkoku*, tout le pays)
3. 国 (*kuni ; koku*, pays)
4. 国都 (*kokuto*, capitale du pays)
5. 都道府県 (*todôfuken*, départements)
6. 都 (*miyako ; to,* capitale ; département de Tôkyô)
7. 都道 (*todô*, route départementale de Tôkyô)
8. 道 (*michi*, chemin)
9. 道府県 (*dôfuken*, départements sauf Tôkyô)
10. 府県 (*fuken*, départements sauf Tôkyô et Hokkaidô)
11. 府 (*fu*, départements de Kyôto et Ôsaka)
12. 県 (*ken*, départements)
13. 県議 (*kengi*, conseil départemental)
14. 県議会 (*kengikai*, conseil départemental)
15. 議会 (*gikai*, parlement)
16. 会 (*kai*, assemblée ; association)
17. 会議 (*kaigi*, réunion)
18. 議長 (*gichô*, président (du conseil))
19. 長 (*chô*, chef)

entraînant ainsi plusieurs possibilités de segmentation comme présenté sur le schéma 2.1.

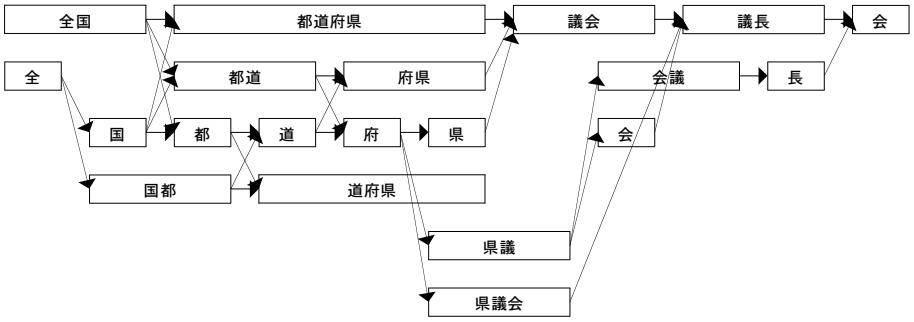


Fig. 2.1 – Multiplicité des segmentations

L'opération *tango bunkatsu* a pour but de trouver la segmentation correcte, en l'occurrence 全国 | 都道府県 | 議会 | 議長 | 会 (Assemblée des présidents de conseil départemental du Japon), parmi ces différentes possibilités. Il existe différentes méthodes permettant de réaliser cette opération, adoptant différents critères pour le choix d'une segmentation.

Nous allons étudier dans les sections suivantes différentes méthodes de segmentation.

2.4 Méthodes basées sur les règles heuristiques

Nous présentons dans cette section les techniques heuristiques de segmentation des phrases en japonais, reconnues efficaces.[2]

2.4.1 Longest Match Method

La première méthode, *longest match method* (最長一致法, *saichô itchi hô*), consiste à trouver chaque *tanjungo*, mot simple, composant la phrase de gauche à droite et lorsqu'il y a plusieurs séquences possibles, c'est l'unité la plus longue qui est choisie. Ne prenant en compte qu'une seule possibilité à chaque choix, elle a comme avantage la rapidité et un besoin peu élevé en mémoire, mais son taux de réussite est peu élevé, à savoir de l'ordre de 80 %. Ce chiffre montre l'insuffisance de cette méthode utilisée seule pour réaliser une segmentation. De plus, elle n'apporte aucune justification sur le choix de l'unité la plus longue.

2.4.2 Méthode du nombre minimum de segments

La deuxième méthode dite du nombre minimum de segments (分割数最小法, *bunkatsu-sû saishô hô*) consiste à trouver la segmentation pour laquelle les unités composantes sont les moins nombreuses. Étant une méthode de recherche exhaustive – i.e. qui prend en compte chaque fois toutes les possibilités –, elle nécessite plus de mémoire que la précédente mais son taux d'erreur est plus faible.

Cependant, cette méthode nécessite également, tout comme les autres méthodes heuristiques, la prise en compte de beaucoup d'exceptions. En outre, elle ne donne pas de solution exacte dans le cas de plusieurs interprétations avec un même nombre de segments.

2.4.3 Méthode de segmentation par type de caractère

Il existe par ailleurs une autre méthode heuristique efficace qui a recours à une des particularités du système d'écriture du japonais utilisant trois types de caractères différents selon la nature des mots. Cette méthode de segmentation par type de caractères (字種切り法, *jishu giri hô*), consiste à réaliser la segmentation en fonction du changement de type de caractère. En effet, en japonais la partie variable des mots variants – comme les verbes ou

[2]Le terme heuristique est défini comme une méthode de résolution des problèmes, non fondée sur un modèle formel et qui n'aboutit pas nécessairement à une solution (arrêté du 27 juin 1989, Journal Officiel du 16 septembre 1989). Cette méthode stratégique indirecte est opposée à l'algorithmique, méthode systématique, qui donne par conséquent des résultats fiables. Mais si le résultat de la méthode heuristique n'est pas garanti, c'est qu'elle explore seulement les possibilités les plus favorables. Elle fait ainsi gagner un temps considérable. Pour la résolution de certains problèmes complexes, l'usage de l'algorithmique est impossible, car elle peut provoquer une « explosion combinatoire ».

les qualificatifs – et les mots dits « vides » n'ayant qu'une fonction grammaticale – tels que les particules –, ont comme particularité d'être représentés en caractères appelés *hiragana*, tandis que les mots « pleins » et les radicaux ayant un sens sont souvent représentés par des idéogrammes appelés *kanji*. Le changement de type de caractère correspond donc à peu près à la frontière entre deux unités comme dans l'exemple ci-dessous.

明日　モンパルナス　で　大学　の　友人　と　食事　する
(demain - Montparnasse - à - université - de - ami - avec - repas (radical) - faire (partie var.))
(Demain, je prendrai un repas avec des amis de l'université à Montparnasse.)

Néanmoins, il existe de nombreuses exceptions. Il n'est pas possible de couper correctement une phrase contenant un/des adverbes constitués uniquement de *kanji*. Ainsi, une phrase telle que :

今回直接会談した

sera segmentée de manière erronée en :

今回直接会談｜した

au lieu de :

今回｜直接｜会談した
(cette fois - directement - s'entretenir (au passé))
(Ils se sont cette fois entretenus directement.)

Les mots contenant des *kana* au milieu de *kanji* posent également des problèmes. Un mot tel que :

申し分ない (sans aucun défaut)

sera segmenté en :

申｜し｜分｜ない

alors qu'il doit être considéré comme une seule unité.

Ainsi le taux de réussite de cette méthode est en réalité assez faible. Toutefois, elle est souvent utilisée pour l'analyse des mots non référencés dans le dictionnaire, permettant de considérer une suite de caractères de même type comme une unité. Par ailleurs, cette méthode basée sur les informations portées par les caractères eux-mêmes, n'est pas utilisable pour le développement d'un programme IME. Pour des applications de ce type, il faut une méthode qui ne présuppose pas les informations calculables par la forme de la phrase à traiter.

Par ailleurs, en ajoutant des informations sur les *bunsetsu* (文節, syntagme), unités syntaxiques du japonais, telles que « la limite entre une particule et un substantif constitue une frontière de *bunsetsu* », on peut réaliser non seulement la segmentation en *tango* mais aussi en *bunsetsu*. Il existe principalement deux méthodes utilisant la segmentation en ce type d'unité [50].

2.4.4 Méthode du nombre minimum de syntagmes

La méthode du nombre minimum de syntagmes (文節数最小法, *bunsetsu-sû saishô hô*) est complètement similaire à la méthode du nombre minimum de segments à la différence près qu'elle utilise non pas les unités lexicales mais les *bunsetsu*. Dans les articles [59, 60], le principe de cette méthode est justifié – contrairement à la *longest match method* où aucune justification de la raison du choix de l'unité la plus longue n'est proposée – par le fait que moins il y a de *bunsetsu* constituant la phrase, plus il est probable que cette structure satisfasse les règles syntaxiques de 係り受け (kakari uke, « dépendance-réception ») qui conditionnent la relation entre les *bunsetsu*. Cette publication démontre l'efficacité de cette méthode avec un taux d'erreur de 7,0 %, à mettre en rapport avec les 12,4 % de la *longest match method*.

2.4.5 Méthode de la plus longue correspondance de deux syntagmes

La méthode de la plus longue correspondance de deux syntagmes (二文節最長一致法, *ni bunsetsu saichô itchi hô*), présentée dans [37], combine la *longest match method* et la segmentation en *bunsetsu*. Cette méthode extrait toutes les possibilités de *bunsetsu* non pas un par un mais deux par deux, et choisit le cas des deux *bunsetsu* consécutifs les plus longs, afin d'éviter les erreurs fréquentes dans le cas où on ne traite chaque fois qu'un seul *bunsetsu* qui peut inclure une partie du *bunsetsu* suivant. Par exemple, l'analyse de la phrase :

そういうざっしを (*so u i u za sshi wo*)

donnera comme possibilités :

- *so u i u* (そういう, « ce type de ») | *za sshi wo* (雑誌を, « magazine (COD) »)
- *so u i* (相違, « différence », 僧衣, « habits de moine », etc.) | **u za* ... (impossible)
- *so u* (そう, « comme ça ») | *i u* (言う, « dire »)

On garde d'abord parmi ces possibilités la première et la troisième qui contiennent deux *bunsetsu* corrects. Ensuite, en comparant la longueur de ces deux candidats, on choisit le plus long, à savoir le premier.

2.5 Méthodes à table de connexion

2.5.1 Table de connexion des catégories

Dans la mesure où il est impossible de trouver les combinaisons correctes uniquement avec des principes heuristiques, il est efficace de vérifier si chaque

connexion entre deux unités est possible ou non. Pour cette opération, on utilise un tableau appelé « table de connexion des catégories » (品詞接続表, *hinshi setsuzoku hyô*) comme présenté ci-dessous à titre d'exemple.

	Substantif	Particule	Qualificatif	Adverbe	Auxiliaire	Verbe
Substantif	1	1	0	0	1	0
Particule	1	0	1	1	0	1
Qualificatif	1	0	0	0	1	0
Adverbe	1	0	0	0	0	1
Auxiliaire	1	0	0	0	1	1
Verbe	1	0	0	0	1	1

Ce tableau définit les possibilités de connexion entre deux unités consécutives. Lorsqu'on vérifie la possibilité de connexion par exemple pour une suite qualificatif – substantif, on regarde la valeur située à l'intersection de la ligne Qualificatif et de la colonne Substantif. Comme elle vaut 1, cette connexion est possible. La valeur 0 signifie l'impossibilité de connexion pour la suite considérée. Voyons comme exemple le cas de la transformation des *kana* en *kanji* avec la phrase :

へんなじがでる (*he n' na ji ga de ru*).

Si le dictionnaire contient les mots :

変な (*hen'na*, « bizarre », qualificatif)
字 (*ji*, « caractère », substantif)
自我 (*ji ga*, « soi », substantif)
が (*ga*, particule)
出る (*de ru*, « apparaître », verbe)

avec les méthodes heuristiques présentées précédemment on obtiendrait le résultat erroné suivant :

変な自我出る

En revanche, avec la table de connexion des catégories qui interdit la connexion directe entre un substantif et un verbe, on peut obtenir le résultat correct :

変な字が出る (Des caractères bizarres apparaissent.)

Cependant, dans le langage proche de la langue parlée, les particules présentes entre le substantif et le verbe sont souvent omises. Il est donc difficile de définir la possibilité de connexion entre deux unités uniquement par deux valeurs 0 et 1. Aussi est-il intéressant d'établir une échelle des possibilités de connexion, ce qui est proposé dans les méthodes basées sur les coûts de composition, présentées dans la section suivante.

2.5.2 Méthode du Coût de Connexion Minimum

Lorsqu'on utilise seulement les méthodes heuristiques et les règles de composition, on obtient souvent des résultats erronés. Comme nous l'avons

vu précédemment, il est difficile de définir une possibilité de connexion entre deux unités uniquement par deux valeurs 0 et 1. La méthode du coût de connexion minimum (接続コスト最小法, *setsuzoku kosuto saishô hô*) définit le degré de possibilité de connexion entre des unités par un coût. Plus la connexion est rare, plus ce coût est élevé. Cette méthode choisit la segmentation dont la somme des coûts est la moins élevée, somme calculée à partir de chacun des coûts de composition des unités constituant la phrase. Elle tient compte également des coûts de chaque mot. Par exemple, 「じ」(*ji*) peut être interprété par différents substantifs comme 「字」 « caractère », 「辞」 « mot vide » ou encore 「痔」 « hémorroïde », mais la fréquence d'apparition de ces mots varient très largement. On définit donc en fonction de leur fréquence un coût qui doit être élevé si leur utilisation est rare, favorisant ainsi les mots les plus utilisés lors du choix dans la segmentation.

Mais l'absence de règle pour déterminer ces coûts représente le principal inconvénient de cette méthode. De plus, les coûts varient souvent d'un domaine à un autre. Nous allons parler dans la section suivante de la segmentation avec un « modèle linguistique fondé sur les statistiques », qui s'appuie sur le même principe que la méthode du coût de connexion minimum, tout en s'affranchissant du problème d'affectation des coûts.

2.5.3 Méthodes d'approche statistique

Ces techniques basées sur le « modèle linguistique fondé sur les statistiques » (統計的言語モデル, *tôkei teki gengo moderu*) utilisent des modèles linguistiques conçus par apprentissage statistique – donc mieux justifiés que l'affectation arbitraire des coûts de la méthode du coût de connexion minimum –, cette approche statistique s'étant largement développée au Japon ces dernières années. Nous étudions un des modèles les plus utilisés : les Modèles de Markov Cachés.

Modèles de Markov Cachés

Avec cette approche, on définit tout d'abord mathématiquement la segmentation.
Soient :

- S, une phrase ; c, un caractère ; m, la longueur de la phrase ;
- W, une suite de mots ; w, un mot ; n, le nombre de mots ;
- T, une suite de catégories ; t, la catégorie attribuée à un mot constituant la suite de mots,

On définit :

- $S = c_1 \dots c_m$
- $W = w_1 \dots w_n$
- $T = t_1 \dots t_n$

Dans ces conditions, le couple d'une segmentation en mots et d'une attribution de catégorie $(\widehat{W}, \widehat{T})$ entraînant la probabilité conjointe la plus grande, de suite de mots et de suite de catégories, $P(W,T)$ est :

$$(\widehat{W}, \widehat{T}) = \arg\max_{W,T} P(W,T/S) = \arg\max_{W,T} P(W,T)$$

Les modèles de probabilité permettant de calculer $P(W,T)$ sont appelés « *word segmentation model* » (単語分割モデル, *tango bunkatsu moderu*). Pour la segmentation du japonais, on utilise en général les Modèles de Markov Cachés. Dans ce modèle, la probabilité conjointe $P(W,T)$ est obtenue par multiplication de la probabilité de composition de deux catégories $P(t_i/t_{i-1})$ et de la probabilité d'apparition d'un mot $P(w_i/t_i)$.

$$P(W,T) = \prod_{i=1}^{n} P(t_i/t_{i-1}) \cdot P(w_i/t_i)$$

Mais il est plus commode de considérer également le début de phrase et la fin de phrase comme des symboles. On utilise souvent la formule suivante, où le symbole # représente la frontière de phrase.

$$P(W,T) = P(t_1/\#) \cdot \prod_{i=2}^{n} P(t_i/t_{i-1}) \cdot P(w_i/t_i) \cdot P(\#/t_n)$$

Nous présentons ci-dessous un exemple de la probabilité de composition de deux catégories et de la probabilité d'apparition d'un mot.

	Substantif	Particule	Qualificatif	Adverbe	Auxiliaire	Verbe
Substantif	0,77	0,04	0,00	0,01	0,16	0,02
Particule	0,84	0,00	0,08	0,04	0,00	0,04
Qualificatif	0,97	0,00	0,00	0,00	0,03	0,00
Adverbe	0,50	0,00	0,02	0,02	0,02	0,44
Auxiliaire	0,48	0,00	0,00	0,00	0,35	0,16
Verbe	0,05	0,00	0,00	0,00	0,88	0,07

Écriture	Lecture	Catégorie	Probabilité
会議	かいぎ	Substantif	0,21
用紙	ようし	Substantif	0,16
が	が	Particule	0,17
送る	おくる	Verbe	0,05
申し込む	もうしこむ	Verbe	0,03
...	...	...	...

En considérant les catégories comme des états et les mots comme des symboles de sortie, on peut représenter la génération de la langue naturelle par un Modèle de Markov Caché. Les Modèles de Markov sont des modèles de probabilité qui supposent que la probabilité de génération d'un symbole

dépend uniquement du symbole précédent. En revanche, les modèles de Markov Cachés sont constitués d'états internes changeant selon un processus de Markov et d'un générateur de symboles dont la répartition de probabilité dépend de ces états. Un modèle de Markov Caché est donc, par définition, un automate d'états limités de probabilité. Ainsi nous pouvons représenter la probabilité de composition de deux catégories et la probabilité d'apparition d'un mot par un automate de Markov Caché comme présenté sur le schéma 2.2.

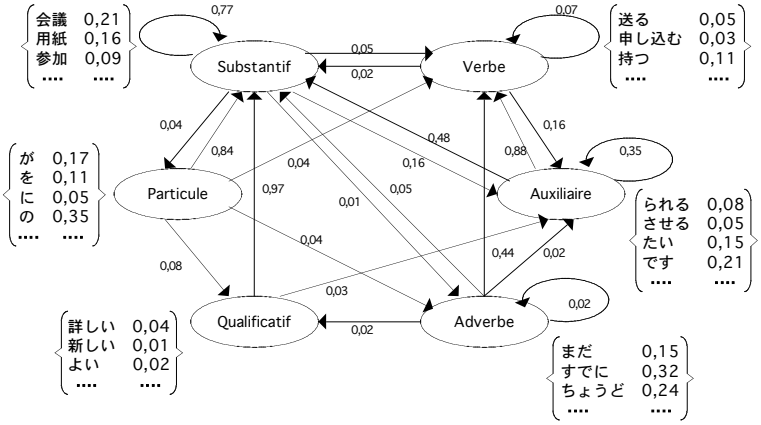


FIG. 2.2 – Automate de Markov Caché

Les paramètres des Modèles de Markov Cachés peuvent être obtenus avec un grand nombre de textes segmentés en unités lexicales et étiquetés à la main, en calculant la fréquence relative de l'objet correspondant.

$$P(t_i/t_{i-1}) = f(t_i/t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

$$P(w_i/t_i) = f(w_i/t_i) = \frac{C(w_i, t_i)}{C(t_i)}$$

$C(\ )$ représente ici la fréquence d'apparition. La probabilité de composition de deux catégories est le rapport entre la fréquence d'apparition des deux catégories consécutives t_{i-1} t_i et la fréquence d'apparition de la catégorie de gauche t_{i-1}. La probabilité d'apparition d'un mot dans une catégorie donnée est le rapport entre la fréquence d'apparition du mot w_i de la catégorie t_i et la fréquence d'apparition de la catégorie t_i.

Les analyseurs morphologiques, JUMAN [35] et CHASEN [40], adoptent comme beaucoup d'autres systèmes, cette approche statistique. Cette méthode très efficace est donc utilisée aussi bien pour les applications de type IME que pour la segmentation des phrases en écritures mélangées. Mais cette situation présente une sorte de contradiction dans la mesure où ces

deux types d'applications ont des données de nature différente : les données du second type d'application ont une forme munie d'informations à explorer, mais qui ne sont pas du tout exploitées.

2.6 Autres algorithmes

On a découvert, assez récemment que l'utilisation d'un algorithme d'analyse syntaxique pour la construction d'un analyseur morphologique était également possible.

Une équipe de l'ICOT[3] a, par exemple, développé un analyseur morphologique basé sur l'algorithme d'analyse syntaxique de CYK. En effet, l'analyse morphologique dans laquelle on vérifie la possibilité de connexion des unités lexicales, comme le font les méthodes présentées précédemment, a un point commun avec le principe de certains types d'analyseurs syntaxiques, tels que l'analyse basée sur l'algorithme CYK, où la possibilité de connexion de deux symboles non terminaux est contrôlée.

Tanaka, chercheur japonais en TAL, signale dans son ouvrage [49] la possibilité de créer un nouvel algorithme d'analyse efficace par l'utilisation d'algorithmes plus généraux comme ceux d'Early ou de Chart, en insistant sur le fait que les algorithmes utilisés actuellement ne sont pas tout à fait satisfaisants pour enlever un grand nombre d'ambiguïtés présentes dans l'analyse morphologique du japonais. Il dit dans son ouvrage :

> « Jusqu'à maintenant, la plupart des méthodes d'analyse morphologique ont adopté un algorithme basé sur l'heuristique, indépendant des algoritmes d'analyse syntaxique. Ce qui est sans doute dû à la séparation totale des étapes morphologique et syntaxique. Cependant, certains chercheurs se sont rendu compte que pour l'analyse morphologique d'une langue n'ayant pas de séparateurs d'unité lexicale telles que le japonais, les informations de niveau syntaxique, sémantique, voire contextuel étaient nécessaires. L'analyse morphologique ne peut pas être traitée comme une procédure totalement séparée des autres étapes. Aussi, faut-il prendre en compte les autres étapes pour réétudier tous les algorithmes développés jusqu'à aujourd'hui. »

[3] `http://www.icot.or.jp`

Chapitre 3

Généralités

3.1 Introduction

Nous allons examiner dans ce chapitre les connaissances de base de l'analyse syntaxique. Après avoir défini cette notion en nous plaçant dans le cadre du traitement automatique des langues, nous étudierons l'analyse syntaxique sous différents angles, avec l'étude de ses applications et de ses outils. Cela nous permettra de poser des bases claires et apportera tout au long du présent document une aide à la compréhension.

3.2 Définitions de l'analyse syntaxique

Il n'est pas aisé de donner une définition stricte et unique de l'analyse syntaxique, car chaque domaine en rapport avec celle-ci l'interprète sous différents angles qui lui sont propres.

Dans les grammaires traditionnelles, on appelle syntaxe la partie de la grammaire décrivant les règles par lesquelles se combinent en phrases les unités significatives. Elle traite des fonctions que les mots peuvent remplir dans la phrase. Par cette définition, l'analyse syntaxique traite des unités syntaxiques, et consiste en l'assignation d'une description syntaxique à ces unités, c'est-à-dire la reconnaissance de la fonction de toutes les unités lexicales constituant cette unité syntaxique et la mise en évidence de rapports entre les unités lexicales.

Dans la théorie des langages formels, on définit l'analyse syntaxique – une des étapes de compilation d'un programme écrit dans un langage de programmation de haut niveau –, par exemple de la façon suivante ([2]) : « *Parsing, or syntax analysis, as it is sometimes known, is a process in which the string of tokens is examined to determine whether the string obeys certain structural conventions explicit in the syntactic definition of the language* ». Il s'agit donc du traitement d'une séquence de caractères comprise entre deux séparateurs de niveau syntaxique et de l'assignation d'une description

structurelle à cette séquence.

Pour le Traitement Automatique des Langues, l'analyse syntaxique concerne les unités syntaxiques, définies comme des séquences de caractères comprises entre deux séparateurs. Elle consiste en l'identification des syntagmes et de leur fonction à l'aide d'un ensemble de règles syntaxiques formalisées et définies antérieurement, appelé grammaire. Elle produit en général comme résultat la représentation sous forme d'arbre de la structure syntaxique globale d'une phrase.

3.3 Types d'analyse syntaxique

Il est possible de découper l'analyse syntaxique selon deux approches, soit en s'intéressant au type même de l'analyse (à savoir partielle ou complète), soit en s'intéressant à la nature de l'analyseur utilisé, robuste ou exigeant. Le type d'analyse syntaxique nécessaire diffère selon la nature de l'application.

L'analyse complète est utilisée par les applications telles que la traduction, alors que l'analyse de type partiel peut suffire pour d'autres comme l'indexation ou l'extraction d'informations.

L'analyse réalisée par un système robuste, qui tolère les fautes, est adaptée à la traduction ou l'indexation de gros volumes de textes. L'analyse avec système « exigeant », détectant toutes les agrammaticalités, est nécessaire pour les correcteurs d'orthographe ou pour les logiciels d'apprentissage d'une langue.

La tendance actuelle est d'utiliser des systèmes d'analyse partielle dits *skimming* ou *shallow parsing*, mais ce type d'analyse partielle nécessite autant de connaissances linguistiques qu'une analyse complète. Il est également fréquent aujourd'hui de découper cette opération d'analyse syntaxique en plusieurs sous-tâches. Cette décomposition permet d'un côté de se fixer un objectif plus raisonnable, mais d'un autre côté ces tâches n'étant pas totalement indépendantes, elle est source de redondance entre les modules.

3.4 Principales applications

Nous allons maintenant passer en revue les principales applications de l'analyse syntaxique qui représentent, comme il a été dit précédemment, la première finalité du domaine du TAL.

Les correcteurs ne tenaient compte au départ que des connaissances lexicales, corrigeant ainsi surtout les fautes de frappe, d'où leur appellation communément répandue de correcteurs d'orthographe. Un correcteur complet nécessiterait en plus non seulement une analyse syntaxique pour les fautes d'accord, mais aussi une étape sémantique pour la correction des homonymes

(ver/verre), ou de certains accords qui supposent d'identifier l'antécédent des pronoms (la forme du vase que j'ai cassé/ée).

L'indexation automatique est une des utilisations les plus importantes de l'analyse syntaxique. C'est entre autres une base élémentaire de l'aide au résumé. Le repérage des groupes nominaux non adverbiaux, opération principale en indexation libre (sans thesaurus), peut se réaliser avec une analyse partielle.

L'interrogation de bases de données est un domaine dans lequel la nécessité d'une véritable analyse syntaxique fut identifiée rapidement. La génération d'une requête suppose en effet l'analyse de la structure de la question, et doit donc être une analyse de type complète.

La simplification de textes est une application récente. La réduction de longueur et de complexité facilite les traitements ultérieurs tels que le résumé ou la traduction. Une analyse complète est préférable mais celle de niveau supra-syntaxique est peu nécessaire.

L'alignement automatique de textes consiste à trouver la correspondance des unités de deux textes de langues différentes et à créer des textes parallèles alignés. Ce traitement présente un meilleur résultat lorsqu'il s'agit du traitement de textes étiquetés syntaxiquement. L'alignement automatique de phrases donne de bons résultats, celui de mots moins, et l'alignement de syntagmes constituerait sans doute un objectif plus réaliste.

L'extraction de connaissances linguistiques à partir de textes consiste à extraire entre autres des termes, des collocations, des cadres de complémentation. Cette opération est plus efficace avec les textes taggés et arborés qu'avec les textes nus.

La génération de phrases accorde également un rôle très important aux connaissances syntaxiques. L'opération de génération suppose d'une part un composant stratégique (*quoi dire ?*) et d'autre part un composant linguistique (*comment le dire ?*) reposant sur les connaissances syntaxiques. Les programmes de génération les plus évolués nécessitent, autant que l'analyse syntaxique automatique, une grammaire électronique, dont la construction voit un essor considérable ces dernières années avec pour objectif la « réutilisabilité » pour de multiples tâches.

3.5 Problème d'ambiguïté

Une des grandes caractéristiques des langues est leur ambiguïté. Une même séquence de lettres peut correspondre à plusieurs mots. Mais ce genre d'ambiguïté provenant de l'homographie et de la polysémie des unités lexicales, cela ne présente pas de réels problèmes pour l'analyse syntaxique.
La phrase

elle s'est acheté une nouvelle serviette

peut être interprétée de deux, ou même trois façons différentes selon le contexte, qui détermine le sens de l'objet acheté, soit serviette de table, soit serviette de toilette, soit petite mallette. Mais la structure syntaxique est identique dans les trois cas.
De même, en japonais, la phrase

はし	を	渡る
pont ; extrémité	*[lieu]*[1]	*v. traverser*

peut être interprétée soit par « *(je) traverse un pont* », soit par « *(je) traverse par l'extrémité* » tout en ayant le même schéma syntaxique.

Cependant, il existe beaucoup de cas où la structure syntaxique est ambiguë. Dans un texte de type brevet, nous rencontrons constamment ce genre d'ambiguïté. Par exemple :

> *Le système comporte un serveur Web, embarqué dans un équipement d'automatisme de l'ensemble d'automatisme, capable de générer des informations statiques ou dynamiques.*

présente une ambiguïté quant à l'attachement du syntagme qualificatif *« capable de générer des informations statiques ou dynamiques »*. La traduction de cette phrase en japonais :

> このシステムはダイナミック又はスタティックなデータが生成できる、コントローラシステムのコントローラ装置に搭載されたWWWサーバを有する。

est également ambiguë de façon analogue.

3.6 Architecture d'un analyseur syntaxique

Un analyseur syntaxique utilise un algorithme d'analyse ainsi qu'un dictionnaire et une grammaire, comme décrit précédemment. Avec ces outils, il produit un arbre représentant la structure syntaxique qui correspond à la structure d'une séquence d'unités lexicales entrées, comme représenté sur la figure 3.1 (voir page suivante), issue de [56].

[1] La traduction entre crochets signifie que l'unité correspondante est un mot grammatical ayant la fonction précisée dans les crochets.

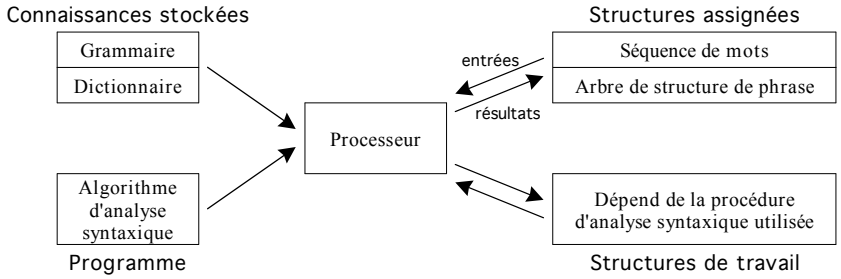


FIG. 3.1 – Connaissances stockées et structures assignées en analyse syntaxique

3.7 Grammaire

Dans cette section, nous nous intéressons à un des éléments constitutifs de l'analyse syntaxique : les grammaires.

Les grammaires pour le TAL nécessitent d'une part les travaux des linguistes en syntaxe comme données, et d'autre part un formalisme qui permet de les représenter de façon cohérente et extensible à l'ensemble de la langue sans se limiter aux seuls exemples cités.

3.7.1 Définitions de la théorie des langages formels

Du fait des contrainte dues à l'implantation informatique, les notions en TAL sont souvent basées sur la théorie des langages formels, donnant ainsi au terme « grammaire » un sens précis et calculatoire, comme pour beaucoup d'autres termes.

Définition 1 (Grammaire)
Une grammaire est un quadruplet $G = (N, \Sigma, P, S)$, où N et Σ sont des alphabets finis et disjoints, N l'ensemble des symboles non-terminaux, Σ l'ensemble des symboles terminaux qui représente l'ensemble des unités lexicales, P l'ensemble des règles de grammaire appelées productions, et $S \in N$ le symbole de départ (ou axiome).

Ainsi, les grammaires sont spécifiées par énumération de leurs productions, avec les productions de l'axiome placées en tête.

Avant de regarder de plus près les grammaires, rappelons les définitions du terme langage et les différentes opérations applicables sur les langages.

Définition 2 (Langage)
Le terme langage dénote un ensemble quelconque de chaînes construites sur un alphabet fixé.

Définition 3 (Opérations sur les langages)
- *Union de L et M, noté $L \cup M$: $L \cup M = \{s \mid s \in L \vee s \in M\}$,*
- *Concaténation de L et M, notée LM : $LM = \{st \mid s \in L \wedge t \in M\}$,*
- *Fermeture de Kleene de L, notée L^* : $L^* = \bigcup_{i=0}^{\infty} L^i$, L^* dénote « un nombre quelconque, éventuellement nul, de concaténations » de L,*
- *Fermeture positive de L, notée L^+ : $L^+ = \bigcup_{i=1}^{\infty} L^i$, L^+ dénote « un nombre quelconque, non nul, de concaténations » de L.*

3.7.2 Production et notion de dérivation

Nous détaillons ici la notion de production, élément constitutif d'une grammaire, ainsi que celle de dérivation, notion étroitement liée à la première.

Considérons la phrase *Pierre chante*. Elle est constituée d'un syntagme nominal suivi d'un syntagme verbal. En utilisant les symboles S pour la phrase, SN pour les syntagmes nominaux et SV pour les syntagmes verbaux, cette règle syntaxique peut s'exprimer comme suit :

$$S \rightarrow SN\ SV$$

où la flèche signifie « peut avoir la forme ». Une telle règle est appelée production. Une production est formée de deux parties, la partie gauche et la partie droite, séparées par une flèche vers la droite. La partie gauche est le nom de la construction syntaxique ; la partie droite donne une forme possible de la construction syntaxique. La partie droite de la production contient des symboles de deux sortes : des symboles terminaux et des symboles non-terminaux.

Une chaîne de symboles terminaux de la partie droite représente une suite éventuellement vide d'unités lexicales. La chaîne de zéro unité lexicale, notée ϵ, est appelée chaîne vide.

Définition 4 (Production)
Une production est représentée sous forme de $\alpha \rightarrow \beta$, où $\alpha \in N^+$, $\beta \in (N \cup \Sigma)^*$.

Les productions dont la partie droite est ϵ, sont appelées productions vides ou ϵ-productions.

Les productions peuvent être interprétées avec le concept de dérivation, qui consiste à engendrer des expressions complexes à partir d'expressions simples. La production $A \rightarrow BA$ est un exemple de dérivation, nous permettant de remplacer une instance quelconque d'un A par BA. Nous pouvons décrire cette action comme $A \Rightarrow BA$, qui se lit « A se dérive en BA ».

Nous disons que $\alpha A \beta \Rightarrow \alpha\gamma\beta$, si $A \rightarrow \gamma$ est une production et α et β sont des chaînes arbitraires de symboles grammaticaux[2].

[2]Les symboles grammaticaux désignent l'ensemble des symboles terminaux et non-terminaux.

Le symbole $\Rightarrow$ représente une dérivation directe et signifie « se dérive en une étape ». Si $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$, on dit que α_1 se dérive en α_n.

La dérivation en zéro, une ou plusieurs étapes est appelée dérivation générale et est représentée par le symbole $\overset{*}{\Rightarrow}$.

Le symbole $\overset{+}{\Rightarrow}$ représente une dérivation non triviale et signifie « se dérive en une ou plusieurs étapes ».

Par ailleurs, dans certaines étapes de dérivation on doit choisir le non-terminal à remplacer. On appelle dérivations gauches les dérivations dans lesquelles le non-terminal le plus à gauche est remplacé à chaque étape. Si $\alpha \Rightarrow \beta$ est une étape dans laquelle le non-terminal le plus à gauche de α est remplacé, on écrit $\alpha \underset{g}{\Rightarrow} \beta$.

Exemple 1 Considérons la grammaire :

$$\begin{aligned} S &\rightarrow xABy \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

La dérivation gauche engendrant la chaîne $xaby$ est :

$$S \underset{g}{\Rightarrow} xABy \underset{g}{\Rightarrow} xaBy \underset{g}{\Rightarrow} xaby$$

De façon analogue, on appelle dérivations droites les dérivations dans lesquelles le non-terminal le plus à droite est remplacé à chaque étape. Les dérivations droites sont parfois appelées dérivations canoniques. Si $\alpha \Rightarrow \beta$ est une étape dans laquelle le non-terminal le plus à droite de α est remplacé, on écrit $\alpha \underset{d}{\Rightarrow} \beta$.

Il résulte de ce que nous venons de voir qu'un langage L ayant une grammaire G peut être défini d'une façon formelle comme suit :

Définition 5 (Langage)
$L(G) = \{u \in \Sigma^* \mid S \overset{*}{\Rightarrow} u\}$

3.7.3 Arbres syntaxiques

Il existe une autre façon d'illustrer la manière dont l'axiome d'une grammaire se dérive en une chaîne du langage. Il s'agit une représentation visuelle à l'aide d'un arbre syntaxique. Si le non-terminal A est défini par la production $A \rightarrow X\ Y\ Z$, alors un arbre syntaxique peut posséder un nœud intérieur étiqueté A et avoir trois fils étiquetés X, Y et Z, de gauche à droite.

Définition 6 (Arbre syntaxique)
Étant doné une grammaire non contextuelle, un arbre syntaxique est un arbre possédant les propriétés suivantes :

1. *La racine est étiquetée par l'axiome.*
2. *Chaque feuille est étiquetée par une unité lexicale ou par ϵ.*
3. *Si A est le non-terminal étiquetant un nœud intérieur et si les étiquettes des fils de ce nœud sont, de gauche à droite, $X_1, X_2, \ldots, X_n$ alors $A \rightarrow X_1\ X_2\ \ldots\ X_n$ est une production. Ici, $X_1, X_2, \ldots, X_n$ représentent soit un non-terminal, soit un terminal. Un cas particulier est $A \rightarrow \epsilon$ qui signifie qu'un nœud étiqueté A a un seul fils étiqueté ϵ.*

Définition 7 (Mot des feuilles)
Les feuilles d'un arbre syntaxique, lues de gauche à droite, constituent le mot des feuilles de l'arbre, qui est la chaîne engendrée ou dérivée à partir du non-terminal situé à la racine de l'arbre syntaxique.

Il est possible de donner une autre définition du langage : c'est l'ensemble des chaînes qui peuvent être engendrées par un arbre syntaxique quelconque. L'analyse d'une chaîne donnée d'unités lexicales est un processus consistant à trouver un arbre syntaxique correspondant à cette chaîne.

3.7.4 Typologie de grammaires

D'un point de vue dérivationnel, une production peut être considérée comme une règle de réécriture dans laquelle le non-terminal en partie gauche est remplacé par la chaîne en partie droite de la production.

Selon le type de règle de réécriture dont chaque grammaire dispose, Chomsky a distingué quatre catégories de grammaire de manière à les hiérarchiser.

Type 0 : grammaire non restreinte,

Type 1 : grammaire contextuelle, dans laquelle la partie droite de production consiste en la partie gauche dont un seul symbole est développé,

Type 2 : grammaire algébrique ou hors contexte (ou encore non contextuelle), dont les productions sont de la forme $A \rightarrow \alpha$, où $A \in N$ et $\alpha \in (N \cup \Sigma)^*$,

Type 3 : grammaire linéaire à droite ou grammaire régulière, dont les productions sont de la forme $A \rightarrow x$, ou $A \rightarrow xB$, où $A \wedge B \in N$ et $x \in \Sigma^*$.

Les grammaire de type 0 sont considérées comme les plus puissantes – plus le chiffre de la catégorie est élevé, moins la grammaire est puissante –, tandis que celles de type 3 sont plus restrictives. Chaque type de grammaire possède une catégorie de langages qui peuvent être définis par une grammaire de même type ou par une grammaire appartenant à une catégorie plus puissante.

Les langages représentés par les grammaires régulières peuvent aussi être représentés par des expressions régulières. Les règles définissant les expressions régulières sur un alphabet Σ sont les suivantes :

Définition 8 (Expressions régulières)

1. ϵ[3] *est une expression régulière qui dénote* $\{\epsilon\}$*, c'est-à-dire l'ensemble constitué de la chaîne vide.*
2. *Si* a *est un symbole de* Σ*, alors* a *est une expression régulière qui dénote* $\{a\}$*, c'est-à-dire l'ensemble constitué de la chaîne* a*.*
3. *Supposons que* r *et* s *soient des expressions régulières dénotant les langages* $L(r)$ *et* $L(s)$*. Alors,*
   a) $(r) \mid (s)$ *est une expression régulière dénotant* $L(r) \cup L(s)$*.*
   b) $(r)(s)$ *est une expression régulière dénotant* $L(r)L(s)$*.*
   c) $(r)^*$ *est une expression régulière dénotant* $(L(r))^*$*.*
   d) (r) *est une expression régulière dénotant* $L(r)$*.*

Les expressions régulières peuvent spécifier la structure des chaînes, telles que les unités lexicales des langages de programmation. L'identificateur de variable en langage C, défini dans [28] comme « une séquence de lettres et de chiffres. Le premier caractère doit être une lettre [...]. Les identificateurs peuvent être de longueur quelconque [...] », peut être représenté par l'expression régulière suivante :

lettre (lettre | chiffre)*

où :

- la barre verticale signifie « ou » ;
- les parenthèses sont utilisées pour grouper des sous-expressions ;
- l'étoile signifie un nombre quelconque, éventuellement nul, d'instances de l'expression parenthésée ;
- la juxtaposition de *lettre* au reste de l'expression signifie la concaténation.

En revanche, beaucoup d'unités syntaxiques ayant une structure récursive, leurs formes ne peuvent pas être spécifiées en utilisant la notation des expressions régulières. Ces constructions peuvent être définies par des grammaires non contextuelles.

3.7.5 Formalismes et Modèles de nouvelles théories linguistiques

Les grammaires à usage humain, très anciennes, n'ayant pour autant jamais été formalisées de façon simple comme une liste ou une table, la question de la représentation des connaissances syntaxiques était et est toujours un problème en soi.

Il convient de distinguer les formalismes simples comme les grammaires de métamorphose, les grammaires de clauses définies (DCG) proposant des métalangages pour les analyses syntaxiques, et les modèles qui proposent une

[3] ϵ représente la chaîne vide, chaîne spéciale de longueur zéro.

théorie linguistique à part entière. On connaît, comme second type de formalisme linguistique plus évolué, la Grammaire lexicale fonctionnelle (LFG), la Grammaire syntagmatique généralisée (GPSG), la Grammaire d'arbres adjoints (TAG) et aussi la Grammaire syntagmatique guidée par les têtes, ou endocentrique (HPSG). Les caractéristiques des grammaires d'unification sont le rejet des transformations, la réhabilitation des descriptions de surface et une formalisation à base de structures de traits.

3.7.6 Objectifs pour la création d'une grammaire

Aucun système ne pourra jamais traiter une langue dans sa totalité. En effet, une grammaire ne peut décrire qu'un sous-ensemble à un moment donné d'une langue : les usages évoluent tout le temps. De plus, la couverture d'une grammaire n'est pas un objectif purement quantitatif. Son accroissement peut produire un facteur de complexité, entraînant ainsi des résultats contre-productifs. Il est également important pour une grammaire d'obtenir la facilité de maintenance et d'interfaçage avec d'autres ressources.

3.8 Outils : Algorithme

Nous allons étudier dans le chapitre 4 page 57 différents algorithmes d'analyse syntaxique. Avant de nous intéresser à ces algorithmes concrèts, nous étudierons les différentes stratégies utilisées pour mieux comprendre leurs différences, et afin de ne pas mélanger des stratégies d'aspects différents, nous les distinguerons d'abord selon leur domaine d'utilisation.

Nous pouvons en premier lieu citer deux stratégies suivant un déroulement vertical pour la construction de l'arbre. Il s'agit des algorithmes ascendant et descendant.

On distingue également deux stratégies entraînant le développement de l'arbre syntaxique sur le plan horizontal. Ces stratégies sont issues de la théorie des graphes, utilisée pour le parcours des graphes.

Il existe également une autre approche dans laquelle on distingue les algorithmes selon leur attitude face au traitement des alternatives. On distingue trois attitudes différentes : les procédures parallèles, les méthodes avec rebroussement (ou retour en arrière) et enfin les méthodes déterministes sans rebroussement.

Nous allons étudier de plus près ces différentes stratégies, afin de définir ensuite des catégories pour ces algorithmes utilisés en analyse syntaxique.

3.8.1 Algorithmes descendant et ascendant

La plupart des méthodes d'analyse tombent dans l'une des deux classes appelées descendante et ascendante. Ces termes font référence à l'ordre suivant lequel sont construits les nœuds de l'arbre syntaxique. Dans les pre-

mières, la construction débute à la racine et descend vers les feuilles, tandis que dans les secondes, la construction débute aux feuilles et remonte vers la racine. Il convient de noter qu'une méthode à la fois descendante et ascendante existe également.

Analyse syntaxique descendante

On effectue la construction descendante d'un arbre syntaxique en partant de la racine, étiquetée par l'axiome, et en réalisant de manière répétitive les deux étapes ci-dessous :

1. Au nœud n, étiqueté par le non-terminal A, choisir une des productions définissant A et construire les fils de n avec les symboles en partie droite de la production.
2. Déterminer le prochain nœud où un sous-arbre doit être construit.

Exemple 2 Considérons la grammaire :

$$S \rightarrow cAd$$
$$A \rightarrow ab$$

et la chaîne d'entrée $w = cabd$.
L'arbre d'analyse pour cette chaîne se construit alors en deux temps, comme représenté sur la figure 3.2.

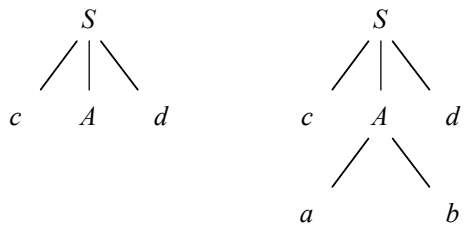


FIG. 3.2 – Étapes d'analyse descendante

Analyse syntaxique ascendante

Il existe un modèle général d'analyse syntaxique ascendante, connu sous le nom d'analyse par décalage-réduction. L'analyse par décalage-réduction a pour but de construire un arbre d'analyse pour une chaîne source en commençant par les feuilles et en remontant vers la racine. Ce processus peut être considéré comme la réduction d'une chaîne w vers l'axiome de la grammaire. À chaque étape de réduction, une sous-chaîne particulière correspondant à la partie droite d'une production est remplacée par le symbole de la partie gauche de cette production et, si la sous-chaîne est choisie correctement à chaque étape, une dérivation droite est ainsi élaborée en sens inverse.

Exemple 3 Considérons la grammaire :

$$S \rightarrow cAd$$
$$A \rightarrow ab$$

La chaîne d'entrée $w = cabd$ peut être réduite vers S par les étapes suivantes :

$$cabd$$
$$cAd$$
$$S$$

Ces réductions élaborent en fait en sens inverse la dérivation suivante :

$$S \Rightarrow cAd \Rightarrow cabd.$$

3.8.2 Recherche en profondeur et recherche en largeur

Ces stratégies sont celles de la théorie des graphes, définissant la façon de parcourir un graphe. La stratégie suivie par un parcours en profondeur d'abord, ou plus simplement en profondeur, est, comme son nom l'indique, de descendre plus profondément dans le graphe chaque fois que c'est possible. L'algorithme de parcours en largeur tient son nom au fait qu'il découvre d'abord tous les sommets situés à une distance k de s avant de découvrir tout sommet situé à la distance $k + 1$.

Recherche en profondeur

Lors d'un parcours en profondeur dans un graphe, les arcs sont explorés à partir du sommet v découvert le plus récemment et dont on n'a pas encore exploré tous les arcs incidents. Lorsque tous les arcs de v ont été explorés, l'algorithme revient en arrière pour explorer les arcs qui partent du sommet à partir duquel v a été découvert. Ce processus se répète jusqu'à ce que tous les sommets accessibles à partir du sommet origine initial aient été découverts. S'il reste des sommets non découverts, on en choisit un qui servira de nouvelle origine, et le parcours reprend à partir de cette origine. Le processus complet est répété jusqu'à ce que tous les sommets aient été découverts.

Dans le cas de l'analyse syntaxique des langues, le symbole le plus à gauche (ou le plus à droite) parmi tous les symboles créés est toujours traité d'abord, jusqu'à ce qu'un symbole terminal ait été atteint (ou, dans le cas d'une analyse ascendante, jusqu'à ce que la racine de l'arbre ait été atteinte). Cette stratégie est raisonnable en combinaison avec une analyse descendante, car les feuilles sont traitées dans l'étape le plus tôt possible pour vérifier ou réfuter la dérivation. Dans l'arbre de la figure 3.3 (voir page suivante), tirée de [22], le côté gauche de la partie A est déjà vérifié par la dérivation des terminaux de t_1 à t_m.

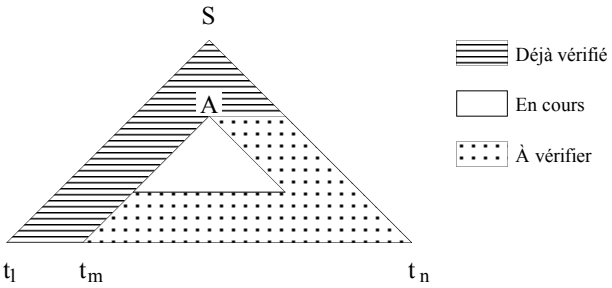


FIG. 3.3 – Recherche en profondeur

Recherche en largeur

Intéressons nous d'abord à la définition du parcours en largeur de la théorie des graphes.

Étant donnés un graphe $G = (S, A)$[4] et un sommet origine s, le parcours en largeur emprunte systématiquement les arcs de G pour découvrir tous les sommets accessibles depuis s. L'algorithme fonctionne aussi bien sur les graphes orientés que sur les graphes non orientés.

Dans le cas de l'analyse syntaxique des langues, les symboles sont traités dans l'ordre de leur création. Par conséquent, l'arbre est rempli dans toute sa largeur. C'est une organisation efficace pour une analyse ascendante car tous les constituants immédiats doivent atteindre le même niveau et doivent être complétés par leurs propres constituants avant de pouvoir être reliés à un plus grand constituant. Une étape d'une telle analyse en largeur est représentée sur la figure 3.4, issue de [22].

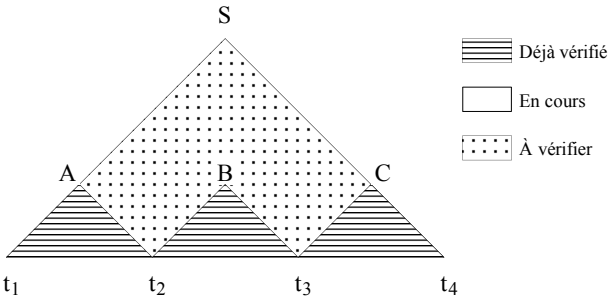


FIG. 3.4 – Recherche en largeur

[4]Un graphe orienté G est représenté par un couple (S, A), où S est un ensemble fini et A est une relation binaire sur S, c'est-à-dire un sous-ensemble du produit cartésien $S \times S$. L'ensemble S est appelé ensemble des sommets de G et A ensemble des arcs de G.

3.8.3 Stratégies pour le traitement des alternatives

Lors du traitement des alternatives, nous avons au moins deux possibilités : tout traiter ou en choisir une seule et la traiter. Le premier est le choix des procédures parallèles, et le second celui des méthodes avec rebroussement, qui peut impliquer des retours arrière. Il existe cependant une autre possibilité avec laquelle tout en traitant une seule alternative, le rebroussement n'est pas nécessaire. C'est la solution réalisée par les méthodes déterministes sans rebroussement.

Méthodes avec rebroussement

En général, la sélection d'une production pour un non-terminal est un processus d'essai-et-erreur. On peut avoir à essayer une production et à rebrousser chemin pour essayer une autre production si la première s'avère ne pas convenir (une production ne convient pas si, après utilisation de cette production, on ne peut pas compléter l'arbre pour que son mot des feuilles corresponde à la chaîne d'entrée). Ainsi cette méthode peut nécessiter des passages répétés sur le texte source, ce qui peut présenter un grand défaut.

De plus, un analyseur basé sur ce type de méthode risque de ne pas pouvoir obtenir le résultat, même s'il existe, car bouclant indéfiniment. Ce problème intervient avec des productions récursives à gauche telles que : $expression \rightarrow expression{+}terme$, où le symbole le plus à gauche de la partie droite est le même que le non-terminal en partie gauche de la production. Toutefois, ce problème n'est pas fatal, car toute récursivité à gauche peut être éliminée d'une grammaire.

Les analyseurs avec rebroussement ne sont pas très populaires. En effet, pour analyser les constructions des langages de programmation, le rebroussement est rarement nécessaire. Dans le cas de l'analyse des langues naturelles, le rebroussement présente une telle perte d'efficacité que ce type d'analyseur est presque inutilisable.

Procédures parallèles

Dans un algorithme de procédures parallèles, lorsqu'un choix entre des alternatives se présente, on tient compte de toutes les alternatives à la fois. Le système doit maintenir une structure de travail qui garde les traces de l'ensemble des états actuels, plutôt qu'un seul comme le font les procédures avec rebroussement. Chaque fois qu'il y a un choix à prendre, est produite une représentation qui couvre la totalité des alternatives.

Tout en résolvant l'inefficacité due au rebroussement, cette méthode a comme défaut presque fatal de nécessiter un espace extrêmement important pour stocker les résultats de tous les calculs de plus en plus nombreux au fur et à mesure que l'étape avance. En réorganisant la structure de travail pour garder séparément la trace des calculs de chaque choix, et en reformulant les

productions, on définit un autre type de méthode qui résout ce problème des méthodes parallèles. Il s'agit d'algorithmes tabulaires, qu'utilisent beaucoup de systèmes d'analyse des langues naturelles.

Méthodes déterministes sans rebroussement

Dans ces méthodes, le symbole de pré-vision (*lookahead*), l'unité lexicale qui vient d'être reconnue dans le flot d'entrée, détermine de manière non ambiguë la procédure à choisir pour chaque non-terminal, ce qui permet d'éviter les rebroussements.

3.8.4 Typologie d'algorithmes d'analyse syntaxique

Les algorithmes d'analyse syntaxique combinent plusieurs stratégies d'aspects différents comme nous venons de le voir. Nous classons ci-dessous les principaux algorithmes selon les stratégies qu'ils ont adoptées.

1. Algorithmes descendants tabulaires : algorithme d'Early, Active Chart Parser.
2. Algorithme descendant déterministe sans rebroussement : algorithme LL(k).
3. Algorithme ascendant tabulaire : algorithme de Cocke-Younger-Kasami.
4. Algorithme bi-directionnel : algorithme du coin gauche.
5. Algorithme ascendant déterministe sans rebroussement : algorithme LR(k).

Nous nous intéresserons dans le prochain chapitre à trois d'entre eux : les algorithmes tabulaires, déterministes sans rebroussement et bidirectionnel.

3.9 Présentation arborescente des résultats

Selon la théorie linguistique adoptée, l'interprétation de la structure syntaxique diffère. Cette différence apparaît clairement dans la représentation sous forme d'arbre de la structure de la phrase. En TAL, deux descriptions structurelles tout à fait différentes sont largement utilisées. Il s'agit des descriptions basées sur la théorie des constituants et sur la théorie de la dépendance. Nous allons voir maintenant comment les représentations de la même phrase diffèrent selon ces théories linguistiques en nous référant aux arbres construits suivant leur théorie respective.

3.9.1 Description structurelle distributionaliste

La notion des constituants est issue d'une théorie générale créée par l'Américain L. Bloomfield ([6]), appelée distributionalisme. L'analyse distributionaliste en constituants immédiats (par abréviation : CI) est une mé-

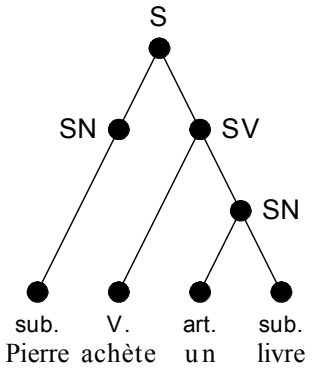


FIG. 3.5 – Représentation distributionaliste

thode de décomposition des énoncés d'un corpus. Elle consiste à attribuer à la phrase une construction hiérarchique, en ce sens qu'elle décompose d'abord l'énoncé en segments, qui sont appelés ses CI, puis subdivise chacun de ceux-ci en sous-segments, qui sont les CI de ce premier CI, et ainsi de suite jusqu'à arriver aux unités minimales.

Ainsi, la phrase *Pierre achète un livre* est d'abord décomposée en deux CI : un SN *Pierre* et un SV *achète un livre*. Ensuite, le SV est décomposé lui-même en deux autres CI : un verbe *achète* et un SN *un livre*. Enfin le SN *un livre* est décomposé à son tour en un article *un* et un substantif *livre*. Cette structure syntaxique est représentée sur le schéma 3.5.

En analyse syntaxique des langues, chaque nœud de l'arbre des constituants correspond à un des segments, plus ou moins grand, de la séquence d'entrée. Les arcs dénotent la relation entre le grand segment et ses constituants.

3.9.2 Description structurelle dépendancielle

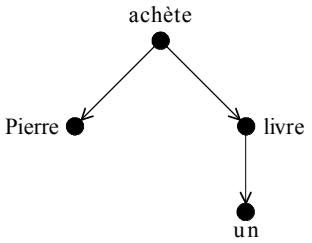


FIG. 3.6 – Représentation dépendancielle

Tout en supprimant l'opposition entre le sujet et le prédicat dans la description structurelle de la phrase, Tesnière [52], linguiste français, a établi la

notion de dépendance. Pour Tesnière, tout complément dépend d'un complété. Décrire les fonctions syntaxiques réalisées dans un énoncé, c'est donc indiquer le réseau de dépendances existant entre les éléments de cet énoncé. Tesnière le représente par une sorte d'arbre, qu'il appelle stemma, où le complément est toujours placé au-dessous du terme complété, et relié à lui par un trait. La figure 3.6 page précédente représente ce que serait le stemma de la phrase *Pierre achète un livre*.

En analyse syntaxique des langues, toutes les unités correspondant à un nœud de cet arbre dépendanciel sont des segments élémentaires de la séquence d'entrée. Les arcs dénotent, comme décrit précédemment, la relation de dépendance entre un segment élémentaire et ses compléments.

Chapitre 4

Algorithmes d'analyse syntaxique

4.1 Introduction

Nous allons maintenant présenter quelques algorithmes d'analyse syntaxique, caractérisés par les stratégies qu'ils utilisent.

Nous nous intéresserons tout d'abord aux algorithmes tabulaires, et nous passerons ensuite à l'étude des algorithmes déterministes sans rebroussement. Enfin, la dernière partie sera consacrée à l'algorithme du coin gauche, algorithme bi-directionnel à la fois descendant et ascendant.

4.2 Algorithmes tabulaires

4.2.1 Introduction : *Well-Formed Substring Tables* ou *Chart*

Comme nous venons de le voir dans le chapitre 3, chaque stratégie a ses avantages et ses inconvénients. Une des inefficacités majeures de la stratégie avec retour en arrière est que la même opération est parfois réalisée à plusieurs reprises via différents chemins. Par exemple, considérons la phrase :

Le garçon avec une scie marche dans les bois

avec la grammaire

1. S → SN SV
2. S → SN SV PREP
3. SN → Dét SN2
4. SN → SN2
5. SN2 → Sub
6. SN2 → Adj SN2
7. SN2 → SN3 PREP
8. SN3 → Sub
9. PREP → SP
10. PREP → SP PREP
11. SP → Prép SN
12. SV → Vb

L'analyse commence par l'application de la production S → SN SV. Après plusieurs étapes d'analyse comprenant aussi des rebroussements, la séquence « Le garçon avec une scie » est analysée comme SN précédant un SV, en l'occurrence « marche ». À cette étape, un autre rebroussement devient nécessaire car il y a encore des unités lexicales à analyser bien qu'avec la production S → SN SV il n'y ait plus de dérivation possible. L'analyseur revient alors en arrière étape par étape en cherchant à trouver d'autres possibilités. Finalement, en remontant jusqu'à l'axiome, on trouve comme nouvelle possibilité la production S → SN SV PREP. L'analyseur recommence à chercher les SN et SV une nouvelle fois, opération déjà réalisée entièrement.

Cette procédure est particulièrement inefficace. Une fois qu'une structure donnée a été reconnue dans une analyse descendante, elle peut être réutilisée à un autre moment, même si l'application d'une production dans laquelle elle est un constituant de la partie droite s'est soldé par un échec. En effet, un des avantages fondamentaux des grammaires hors contexte est le caractère indépendant de leurs productions. Une production « SN → Adj SN2 » peut être appliquée sans se préoccuper de la façon dont SN est utilisé dans d'autres structures. De nombreux analyseurs possèdent des mécanismes de stockage connus sous le nom de *Well-Formed Substring Tables* (Tables de sous-chaînes bien formées) ou *Chart* pour conserver les traces des non-terminaux qui ont été développés à un moment donné de l'analyse, mais qui peuvent être utilisés par d'autres productions.

Un chart peut être visualisé comme un réseau de nœuds représentant les positions dans la phrase, reliés par des arcs représentant les constituants. Un arc peut représenter un constituant de différents niveaux de la phrase, avec pour origine et pour terminaison les nœuds qu'il connecte. Initialement, le chart contient seulement les arcs correspondant aux unités lexicales et à leurs catégories lexicales, comme représenté sur la figure 4.1.

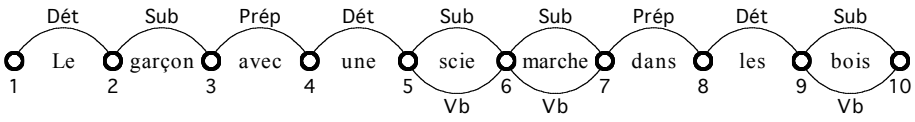


FIG. 4.1 – Chart initialisé pour une analyse

Lorsqu'un constituant d'un niveau donné de la phrase est reconnu, il est ajouté au chart. Au moment où l'analyse de l'exemple précédent passe à l'étape de l'application de la seconde règle de l'axiome, l'état du chart devient alors celui de la figure 4.2 (voir page suivante).

L'analyseur utilise le chart d'une part pour enregistrer les nouveaux arcs représentant les non-terminaux de la partie gauche d'une production appliquée quand sa partie droite est entièrement développée, et d'autre part pour trouver l'arc représentant le non-terminal qui se trouve dans la partie droite

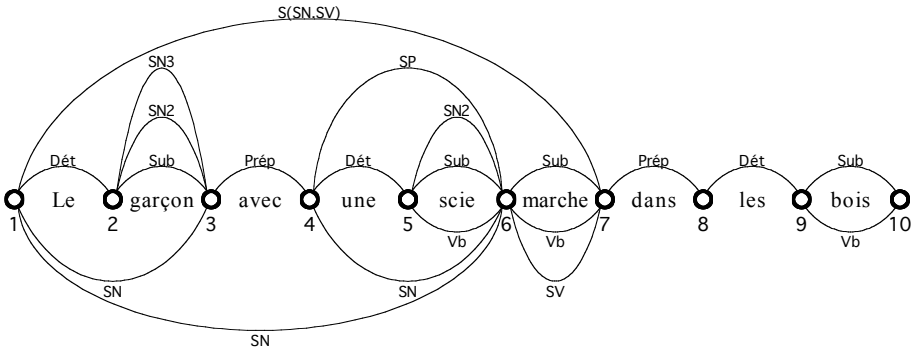


FIG. 4.2 – Chart contenant les enregistrements des constituants trouvés

de la production appliquée et que l'analyseur cherche à développer. Si l'arc correspondant au symbole existe sur le point courant dans le chart, l'analyseur peut avancer le pointeur dans la phrase jusqu'au nœud d'arrivée de l'arc, sans avoir besoin de développer une nouvelle fois le non-terminal en question. Ce qui résout en grande partie le problème d'inefficacité de l'analyse descendante.

Aujourd'hui, nous connaissons des méthodes permettant de conserver plus d'informations que dans la figure 4.2. Le chart décrit jusqu'à présent ne peut conserver que les traces des non-terminaux que l'analyseur a réussi à développer, et ce sans indiquer la production utilisée. Afin de conserver la totalité des opérations réalisées, par exemple, l'*Active Chart Parser* introduit dans le chart, non seulement l'ensemble des arcs dit « complets » représentant un non-terminal reconnu, mais aussi l'ensemble des arcs dits « actifs » représentant un état au cours du développement d'un non-terminal.

Outre l'*Active Chart Parser* qui est un algorithme d'analyse syntaxique descendante, il existe un autre algorithme de chart à stratégie ascendante : le *Bottom-up Chart Parsing*. Nous allons maintenant étudier plus précisément cet algorithme comme exemple concret des algorithmes de chart.

4.2.2 *Bottom-up Chart Parsing*

Bottom-up Chart Parsing est un algorithme d'analyse syntaxique ascendante et parallèle. L'algorithme de *Bottom-up Chart Parsing* est caractérisé par le fait qu'il autorise la règle de génération d'une chaîne vide et qu'il ne nécessite pas l'utilisation d'une grammaire de forme normale Chomsky. Il existe deux types d'algorithme *Bottom-up Chart Parsing* avec des stratégies de recherche en profondeur et en largeur.

Avec cet algorithme, on attribue d'abord entre deux mots consécutifs un numéro appelé **nœud**.

mot1 *mot2* *mot3*
0 1 2 3

On installe ensuite des arcs entre ces nœuds. On crée d'abord les arcs correspondant à chaque unité lexicale, *mot1*, *mot2* et *mot3*, étiquetés par leur catégorie lexicale, *cat1*, *cat2* et *cat3*. On installe ensuite d'autres arcs puis on les étiquette de la façon présentée dans la section 4.2.4 (voir page suivante) en appliquant les productions définies par la grammaire. Ces arcs sont distingués selon leur étiquette en deux types : **arc actif** et **arc non actif**. Le premier contient un/des termes non définis. Par exemple, en appliquant la règle :

$$\text{A} \rightarrow \textit{cat1}$$

le nouvel arc reliant les nœuds 0 et 1 est étiqueté comme [A *cat1*]. En revanche, si on applique les deux règles :

$$\text{A} \rightarrow \textit{cat1}$$
$$\text{C} \rightarrow \text{A B}$$

on obtient un arc qui a comme étiquette [C [A *cat1*][B ?]]. Le symbole « ? » représente le fait que la structure n'est pas encore définie, et il est appelé **terme non défini**. Un arc ayant une telle étiquette qui contient au moins un terme non défini est appelé **arc actif**. L'arc du premier exemple est donc un **arc non actif**. Le terme non défini apparaissant en premier à gauche est appelé **terme non défini le plus à gauche**. L'analyse avance en remplissant la structure de ce terme non défini le plus à gauche. Dans le cas de l'exemple ci-dessus, la structure non définie est remplie par l'arc non actif [B *cat2*] créé en appliquant la règle B → *cat2*, produisant ainsi un nouvel arc non actif ayant pour structure [C [A *cat1*][B *cat2*]].

Il existe deux types d'algorithmes de *Bottom-up Chart Parsing* : en profondeur d'abord, qui réalise toutes les opérations possibles pour une unité avant de passer à l'unité suivante ; en largeur d'abord, qui réalise une opération de même niveau pour toutes les unités avant de passer aux opérations de niveau supérieur. Le deuxième type présente un parallélisme plus élevé que le premier, ce qui lui donne comme avantage une réduction du nombre d'étapes. En revanche le premier avançant l'analyse mot par mot, il facilite le contrôle de la procédure d'analyse et permet d'introduire des moyens d'optimisation de vitesse d'exécution au moment adéquat. Pratt a ainsi créé un algorithme combinant ce premier type d'algorithme de *Bottom-up Chart Parsing* et une matrice d'accessibilité pour réaliser le système LINGOL. Ce système a d'ailleurs été étendu par un chercheur japonais (Unemi), avec l'ajout de la possibilité d'utilisation d'une grammaire hors contexte.

Dans la mesure où l'ordre de traitement des unités varie selon la stratégie (en profondeur d'abord ou en largeur d'abord), l'ordre d'installation des arcs et même le nombre d'arcs installés diffèrent de l'une à l'autre. Cependant

cela ne signifie pas que le principe même de l'algorithme change. Dans la présente étude, nous ne présenterons que l'algorithme de Chart avec recherche en profondeur, mais l'adaptation à la stratégie de recherche en largeur est facilement réalisable à partir du mécanisme que nous allons maintenant étudier.

4.2.3 Définitions

Soit $G =< N, \Sigma, P, S >$ une grammaire hors contexte.
On appelle $A \rightarrow \alpha \bullet \beta$ une production divisée de G, où α est la partie déjà reconnue et β est la partie encore à reconnaître. On utilise des productions divisées pour représenter un arc.

Soit P = $\{A \rightarrow \alpha \bullet \beta \mid A \rightarrow \alpha\beta \in$ P, $\alpha, \beta \in (N \cup \Sigma)^*\}$ l'ensemble des productions divisées de G.

Si $\beta = \varepsilon$, l'arc représenté par A $\rightarrow \alpha \bullet \beta$ est un arc non actif (ou passif).
Si $\beta \neq \varepsilon$, l'arc représenté par A $\rightarrow \alpha \bullet \beta$ est un arc actif.
α s'appelle la partie fermée et β s'appelle la partie ouverte.

4.2.4 Algorithme

[1] Initialisation du chart

tant que $k <$ dernier

faire pour le mot w_k situé entre les nœuds k et $k+1$.

si la catégorie lexicale de w_k = C

alors installation entre les nœuds k et $k+1$ d'un arc non actif ayant comme étiquette [C w_k]

[2] Analyse

tant que $k <$ dernier

répéter les opérations de (i) à (iii) pour tous les arcs non actifs situés entre les nœuds j et $k+1$ ($j < k+1$) ayant comme étiquette [X...]. Pas d'installation de plusieurs arcs ayant la même étiquette.

(i) **si** il existe la règle B $\rightarrow$ X

alors installation entre les nœuds j et $k+1$ d'un arc non actif ayant comme étiquette [B [X...]].

(ii) **si** il existe la règle B $\rightarrow$ X Y ... Z

alors installation entre les nœuds j et $k+1$ d'un arc actif ayant comme étiquette [B [X...][Y ?]...[Z ?]].

(iii) **si** il existe un/des arcs actifs entre les nœuds i et j ayant, dans leur étiquette A, comme terme non défini le plus à gauche [X ?]

alors installation entre les nœuds i et $k+1$ d'un arc actif ayant comme étiquette A' créée à partir de l'étiquette A dont le terme non défini le plus à gauche [X ?] est remplacé par [X...].

[3] Résultat

si il existe au moins un arc non actif reliant le nœud 0 et le dernier nœud ayant comme étiquette [S...]

alors l'analyse est réussie.

sinon l'analyse a échoué.

4.2.5 Exemple

Afin d'éclaircir l'algorithme ci-dessus, voyons à présent son déroulement étape par étape, avec l'exemple ci-dessous :

あかい　と　走る
0　1　2　3

Soit la grammaire suivante :

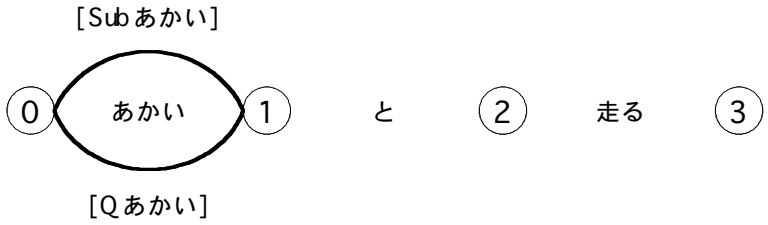
1. S → SP S
2. S → SP V
3. SP → Sub P
4. SP → Q P
5. SP → V P

Initialisation du chart

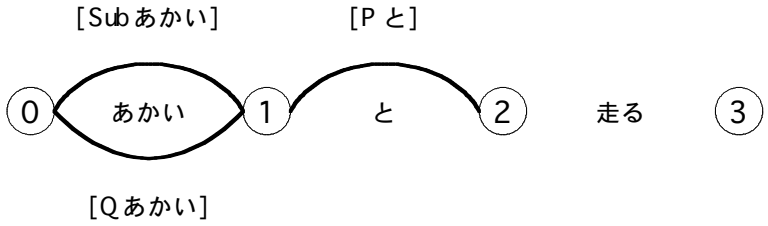
a) Installation de l'arc non actif entre les nœuds 0 et 1, représentant la première unité lexicale あかい, et étiqueté par sa catégorie lexicale [Subあかい[1]].

b) Installation de l'arc non actif entre les nœuds 0 et 1, représentant la première unité lexicale あかい, et étiqueté par sa catégorie lexicale [Qあかい[2]].

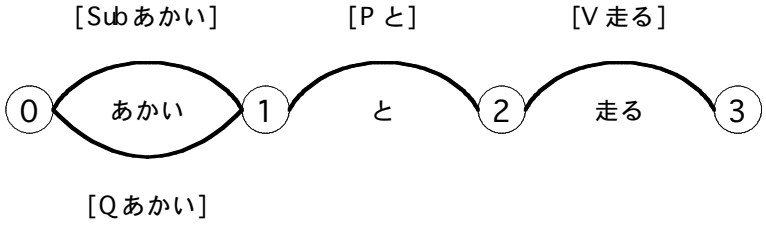
[1] あかい : lecture = *akai*, sens = *Akai* (nom de famille)
[2] あかい : 赤い, lecture = *akai*, sens = rouge



c) Installation de l'arc non actif entre les nœuds 1 et 2, représentant la deuxième unité lexicale と, et étiqueté par sa catégorie lexicale [P と[3]].



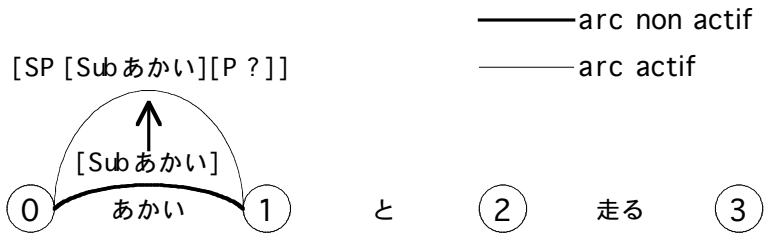
d) Installation de l'arc non actif entre les nœuds 2 et 3, représentant la troisième unité lexicale 走る, et étiqueté par sa catégorie lexicale [V走る[4]] .



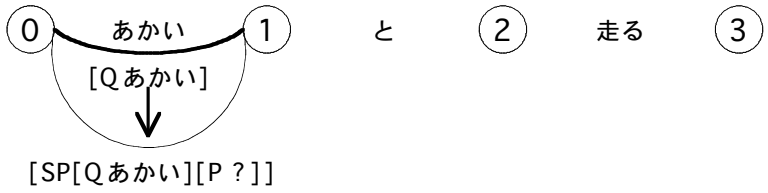
Analyse

e) À partir de l'arc (a) et de la règle 3, installation de l'arc actif entre les nœuds 0 et 1 ayant comme étiquette [SP [Subあかい] [P ?]].

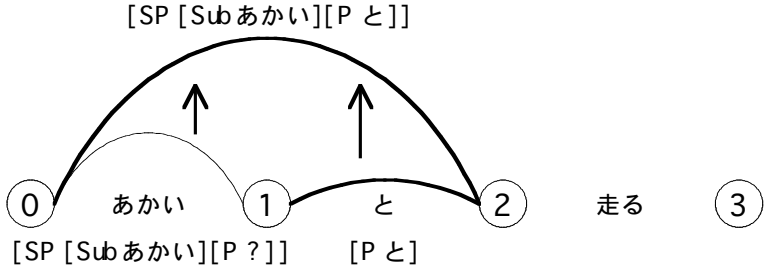
[3] と : lecture = *to*, sens = lorsque ; avec
[4] 走る : lecture = *hashiru*, sens = courir



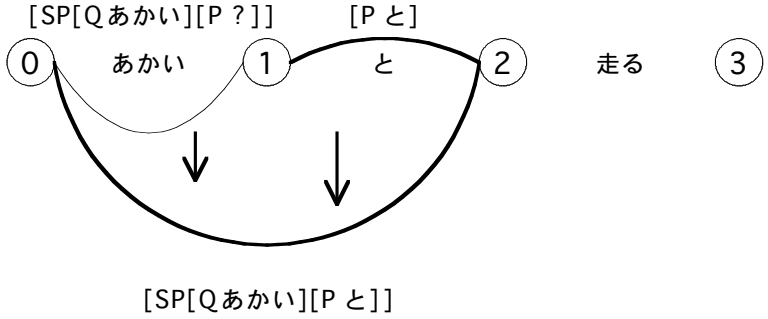
f) À partir de l'arc (b) et de la règle 4, installation de l'arc actif entre les nœuds 0 et 1 ayant comme étiquette [SP [Qあかい] [P ?]].



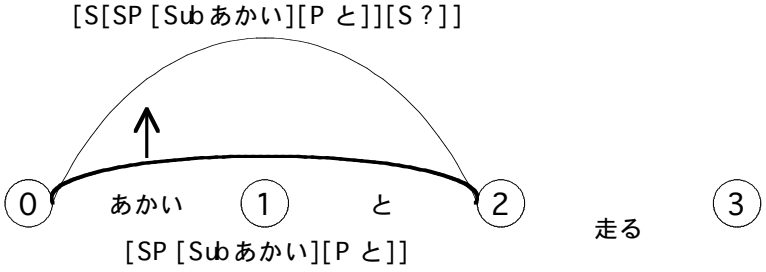
g) À partir des arcs (c) et (e), installation de l'arc non actif entre les nœuds 0 et 2 ayant comme étiquette [SP [Subあかい] [Pと]].



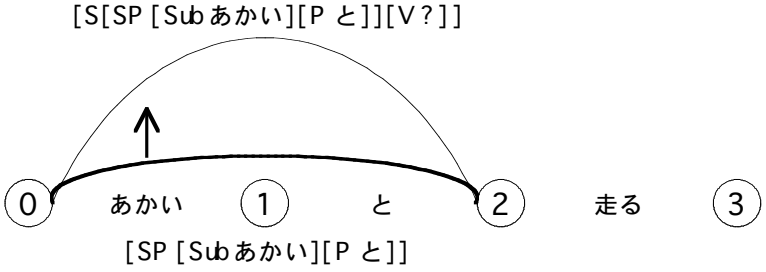
h) À partir des arcs (c) et (f), installation de l'arc non actif entre les nœuds 0 et 2 ayant comme étiquette [SP [Qあかい] [Pと]].



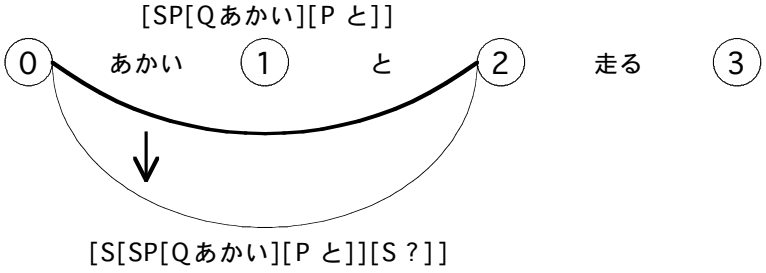
i) À partir de l'arc (g) et de la règle 1, installation de l'arc actif entre les nœuds 0 et 2 ayant comme étiquette [S [SP [Subあかい] [Pと]] [S ?]].



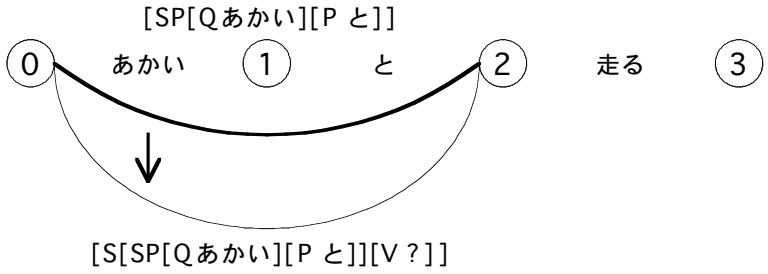
j) À partir de l'arc (g) et de la règle 2, installation de l'arc actif entre les nœuds 0 et 2 ayant comme étiquette [S [SP [Subあかい] [Pと]] [V ?]].



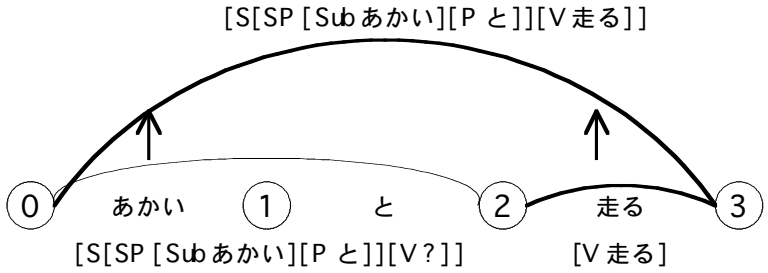
k) À partir de l'arc (h) et de la règle 1, installation de l'arc actif entre les nœuds 0 et 2 ayant comme étiquette [S [SP [Qあかい] [Pと]] [S ?]].



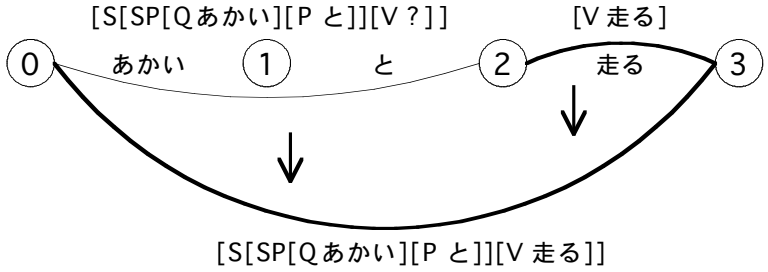
l) À partir de l'arc (h) et de la règle 2, installation de l'arc actif entre les nœuds 0 et 2 ayant comme étiquette [S [SP [Qあかい] [Pと]] [V ?]].



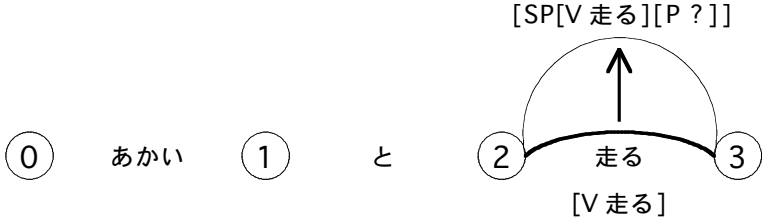
m) À partir des arcs (j) et (d), installation de l'arc actif entre les nœuds 0 et 3 ayant comme étiquette [S [SP [Subあかい] [P ?]] [V走る]].



n) À partir de l'arc (l) et (d), installation de l'arc actif entre les nœuds 0 et 1 ayant comme étiquette [S [SP [Qあかい] [P ?]] [V走る]].

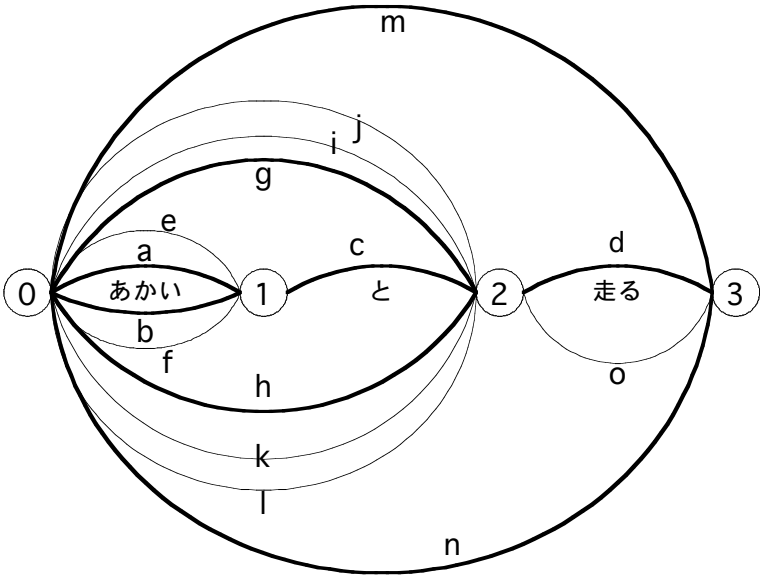


o) À partir de l'arc (d) et de la règle 5, installation de l'arc actif entre les nœuds 2 et 3 ayant comme étiquette [SP [V走る] [P ?]].



Après toutes ces opérations on obtient deux arcs non actifs reliant la tête et la fin de phrase ayant comme étiquette [S...], ce qui signifie la réussite de l'analyse avec deux possibilités de réponse. D'ailleurs, cette obtention de toutes les réponses à la fin de l'analyse montre la stratégie parallèle de cet algorithme.

Si on représente maintenant tous les arcs sur un même schéma, on obtient :



Avec la version « en largeur d'abord » de l'algorithme nous obtiendrions le même schéma, seul l'ordre de création des arcs différant. Par exemple, dans le cas de l'exemple étudié, avec la stratégie de recherche en profondeur, les arcs se produisent dans l'ordre :

$$\{a, b, c, d\} \rightarrow \{e, f\} \rightarrow \{g, h\} \rightarrow \{i, j, k, l\} \rightarrow \{m, n\} \rightarrow \{o\}$$

tandis que, avec l'algorithme avec recherche en largeur, les arcs se créent dans l'ordre :

$$\{a, b, c, d\} \rightarrow \{e, f, o\} \rightarrow \{g, h\} \rightarrow \{i, j, k, l\} \rightarrow \{m, n\}$$[5]

À noter qu'il existe un algorithme améliorant la vitesse d'exécution de l'algorithme de *Bottom-up Chart Parsing* : l'algorithme de Pratt et Unemi, déjà mentionné plus haut.

[5] {} représente la possibilité d'exécution parallèle des opérations.

4.3 Algorithmes déterministes sans rebroussement

4.3.1 Introduction

Il existe deux principaux algorithmes déterministes sans rebroussement : l'algorithme LL et l'algorithme LR.

L'algorithme LL(1) est un algorithme d'analyse déterministe descendante, qui réalise des analyses dites prédictives. Le premier « L » signifie « parcours de l'entrée de la gauche vers la droite » (***L**eft to right scanning of the input*), le second « L » « dérivation gauche » (*constructing a **L**eftmost derivation*), et le 1 indique qu'on utilise un seul symbole d'entrée de pré-vision à chaque étape nécessitant une prise de décision dans l'action d'analyse.

Les grammaires permettant d'appliquer l'algorithme LL(1) sont appelées grammaires LL(1). La principale difficulté dans l'utilisation d'une analyse prédictive réside dans l'écriture d'une grammaire permettant de construire un analyseur prédictif. Bien que l'élimination des récursivités à gauche et la factorisation à gauche soient aisées à réaliser, elles rendent la grammaire résultante difficile à lire et à utiliser.

L'algorithme LR est un algorithme d'analyse syntaxique ascendante. C'est une technique efficace qui peut être utilisée pour analyser une large classe de grammaires non contextuelles. C'est également une méthode très générale d'analyse par décalage-réduction, qui est utilisée dans un grand nombre de constructeurs automatiques d'analyseurs syntaxiques. « L » signifie « parcours de l'entrée de la gauche vers la droite » (***L**eft to right scanning of the input*), « R » signifie « en construisant une dérivation droite inverse » (*constructing a **R**ightmost derivation in reverse*). On parle souvent de l'algorithme LR(k), où k indique le nombre de symboles de pré-vision utilisés pour prendre les décisions d'analyse. Lorsque (k) est omis, il est supposé être égal à 1.

À la différence de l'algorithme LL, la méthode LR a donc comme avantage la disponibilité du constructeur d'analyseurs, qui évite à l'utilisateur une quantité trop importante de travail pour construire à la main cet analyseur pour la grammaire d'une langue. En effet, celui-ci n'a qu'à écrire une grammaire non contextuelle et la soumettre au constructeur afin qu'il produise automatiquement l'analyseur correspondant. Avec ces constructeurs d'analyseurs LR, on peut obtenir automatiquement tous les bénéfices de l'analyse prédictive.

Outre cet avantage, cette méthode présente un certain nombre de points forts, comme par exemple, le fait que tout en étant la méthode par décalage-réduction sans rebroussement la plus globale, elle puisse être implantée aussi efficacement que les autres méthodes par décalage-réduction.

Nous allons présenter l'analyse syntaxique LR comme exemple des algorithmes déterministes sans rebroussement. Mais avant d'entrer dans l'étude du fonctionnement d'un analyseur LR, nous allons nous intéresser à l'analyse

par décalage-réduction en général. Nous étudierons ensuite les techniques de construction des tables d'analyse LR pour une grammaire.

4.3.2 Principe de l'analyse par décalage-réduction

Nous étudions ici plus particulièrement une implantation efficace de l'analyse par décalage-réduction, qui utilise une pile pour conserver les symboles grammaticaux et un tampon d'entrée qui contient la chaîne w à analyser.

Avant d'étudier le principe de la méthode, définissons quelques notions nouvelles.

Définition 9 (Proto-phrase)
Soient G une grammaire et S son axiome.
Si $S \overset{*}{\Rightarrow} \alpha$ où α peut contenir certains non-terminaux, on dit que α est une proto-phrase de G. On appelle proto-phrases gauches les proto-phrases obtenues par les dérivations gauches, et celles qui sont obtenues par les dérivations droites sont appelées proto-phrases droites.

Par cette définition, une phrase est donc une proto-phrase sans non-terminal.

Définition 10 (Manche)
Un manche de proto-phrase droite γ est une production $A \rightarrow \beta$ et une position dans γ où la chaîne β peut être trouvée et remplacée par A pour produire la proto-phrase droite précédente dans une dérivation droite de γ.

Exemple 4 Considérons les dérivations droites suivantes :

$$S \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha \beta w$$

La production $A \rightarrow \beta$ dans la position suivant α est un manche de $\alpha\beta w$.

Si une grammaire est non ambiguë, chaque proto-phrase droite de cette grammaire a exactement un manche. Quelque fois nous disons juste « la sous-chaîne β est un manche pour $\alpha\beta w$ » si la position de β et la production $A \rightarrow \beta$ que nous avons en tête sont claires.

Algorithme

Bien que les opérations de base de l'analyseur soient décaler et réduire, une analyse par décalage-réduction est constituée en fait de quatre actions possibles : (1) décaler, (2) réduire, (3) accepter et (4) erreur.

Nous utilisons le symbole \$ pour marquer à la fois le fond de la pile et l'extrémité droite du tampon d'entrée.

État initial
Pile : \$

Tampon d'entrée : w \$

[1] positionner le pointeur *ps* sur le premier symbole de w\$

[2] répéter indéfiniment

(a) décaler le symbole pointé par *ps* sur la pile /* décaler */

(b) avancer *ps* sur le prochain symbole en entrée

(c) répéter indéfiniment (i) et (ii)

(i) **si** la chaîne de la pile se termine par le manche γ, qui peut se réduire par A,
alors remplacer γ par A sur la pile /* réduire */

(ii) **sinon** arrêter de répéter

(d) **si** *ps* pointe sur \$,
alors arrêter de répéter

[3] **si** l'état de la pile est \$$S$,
alors retourner /* accepter */

[4] **sinon** signaler l'erreur /* erreur */

Exemple 5 Considérons la chaîne d'entrée w :

あかい　と　走る

avec la grammaire :

1. $S \rightarrow SP\ SV$
2. $SP \rightarrow N\ P$
3. $SV \rightarrow V$
4. $N \rightarrow$ あかい
5. $P \rightarrow$ と
6. $V \rightarrow$ 走る

Initialement, la pile est vide, la chaîne w est dans le tampon, et le pointeur pointe sur あかい.

\$　　あかい　と　走る \$

On décale l'unité pointée sur la pile, et on avance *ps* sur le prochain symbole en entrée : と.

\$ あかい　　と　走る \$

On réduit あかい par la production 4.

\$ N　　と　走る \$

On décale l'unité pointée sur la pile, et on avance *ps* sur le prochain symbole en entrée : 走る.

\$ N と　　走る \$

On réduit と par la production 5.

\$ $N\ P$　　走る \$

On réduit $N\ P$ par la production 2.

$$\$\ SP \qquad \overset{\Downarrow}{走る}\ \$$$

On décale l'unité pointée sur la pile, et on avance *ps* sur le prochain symbole en entrée : \$.

$$\$\ SP\ 走る \qquad \overset{\Downarrow}{\$}$$

On réduit 走る par la production 6.

$$\$\ SP\ V \qquad \overset{\Downarrow}{\$}$$

On réduit V par la production 3.

$$\$\ SP\ SV \qquad \overset{\Downarrow}{\$}$$

On réduit $SP\ SV$ par la production 1.

$$\$\ S \qquad \overset{\Downarrow}{\$}$$

L'analyseur s'arrête et annonce la réussite finale de l'analyse.

4.3.3 Architecture d'analyse LR

Nous étudions à présent le principe de l'analyse syntaxique LR, qui est une sous-classe d'analyse par décalage-réduction.

Un modèle d'analyseur LR est schématisé sur la figure 4.3.

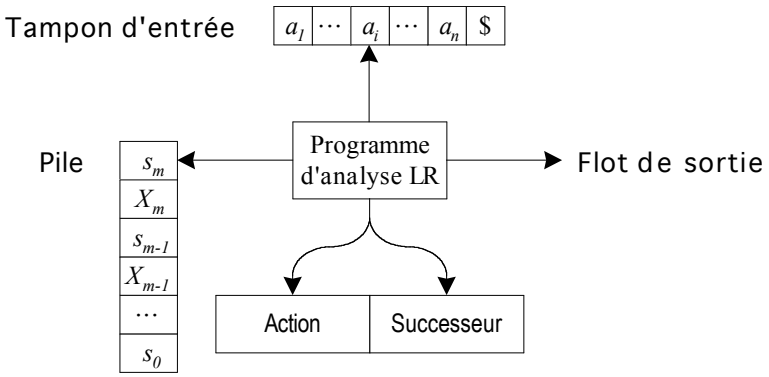


FIG. 4.3 – Modèle d'un analyseur LR

Il consiste en un tampon d'entrée, un flot de sortie, une pile, un programme pilote et des tables d'analyse subdivisées en deux parties (Action et Successeur). Le programme moteur est le même pour tous les analyseurs LR, seules les tables d'analyse changeant d'un analyseur à l'autre. Le programme d'analyse lit les unités lexicales les unes après les autres dans le tampon d'entrée. Il utilise une pile pour y ranger des chaînes de la forme $s_0\ X_1 s_1\ X_2 s_2 \ldots X_m s_m$, où s_m est au sommet, chaque X_i est un symbole de la grammaire et chaque s_i est un symbole appelé *état*. Chaque état résume l'information contenue dans la pile en dessous de lui ; la combinaison du numéro de l'état en sommet de pile et du symbole d'entrée courant est utilisée

pour indicer les tables et déterminer l'action d'analyse, `décaler` ou `réduire`, à effectuer.

4.3.4 Programme moteur

Algoritme

Données : une chaîne d'entrée w et des tables d'analyse LR (fonctions Action et Successeur) pour une grammaire G.
Résultat : si w est dans $L(G)$, une analyse ascendante de w ; sinon, une indication d'erreur.
État initial :
Pile : s_0, où s_0 est l'état initial,
Tampon d'entrée : w\$.

[1] positionner le pointeur ps sur le premier symbole de w\$

[2] répéter indéfiniment

(a) soit s l'état en sommet de pile et a le symbole pointé par ps

(b) **si** Action$[s, a]$ = décaler s',
alors

(i) empiler a puis s'

(ii) avancer ps sur le prochain symbole en entrée

(c) **si** Action$[s, a]$ = réduire par $A \to \beta$,
alors

(i) dépiler $2 \times |\beta|$ symboles

(ii) soit s' le nouvel état sommet de pile

(iii) empiler A puis Successeur$[s', A]$

(iv) émettre en sortie une identification de la production $A \to \beta$

(d) **sinon si** Action$[s, a]$ = accepter,
alors fin d'analyse

(e) **sinon** Erreur()

4.3.5 Grammaire LR

Il y a des grammaires non contextuelles pour lesquelles l'analyse par décalage-réduction ne peut pas être utilisée. Ces grammaires peuvent entraîner une configuration dans laquelle l'analyseur par décalage-réduction ne peut pas décider, connaissant le contenu de la pile d'analyse tout entière et les prochains symboles en entrée, s'il doit décaler ou réduire (**conflit décaler / réduire**) ou quelle réduction choisir parmi plusieurs (**conflit réduire / réduire**). Elles ne font pas partie de la classe des grammaires LR.

On appelle grammaire LR une grammaire pour laquelle il est possible de construire des tables d'analyse. Une grammaire ambiguë, par exemple, qui peut avoir plus d'un arbre syntaxique pour une chaîne donnée d'unités lexicales, ne peut jamais être LR. Il y a cependant des moyens pour les rendre LR. Pour qu'une grammaire soit LR, il suffit qu'un analyseur par décalage-réduction gauche-droite soit capable de reconnaître les manches quand ils apparaissent en sommet de pile.

4.3.6 Méthodes pour la construction des tables d'analyse

Il existe trois principales techniques pour construire les tables d'analyse pour une grammaire. Elles diffèrent par leur facilité d'implantation et leur puissance.

La première méthode, appelée « Simple LR » (SLR en abrégé), est la plus facile à implanter, mais la moins puissante des trois en termes du nombre de grammaires pour lesquelles elle réussit. En effet, pour certaines grammaires, la production des tables d'analyse peut échouer alors qu'elle réussirait avec les autres méthodes (les deux autres méthodes complètent la méthode SLR avec des informations de pré-vision). Les tables d'analyse obtenues par cette méthode sont appelées tables SLR. On appelle un analyseur utilisant ces tables analyseur SLR, et enfin une grammaire pour laquelle il est possible de construire un analyseur SLR est dite grammaire SLR.

La deuxième méthode, appelée LR canonique, est la plus puissante mais aussi la plus coûteuse. La troisième méthode, appelée « LookAhead LR » (LALR en abrégé), a une puissance et un coût intermédiaires entre les deux autres. À titre de comparaison concernant la taille des analyseurs, les tables SLR et LALR d'une grammaire ont le même nombre d'états, et ce nombre est de plusieurs centaines pour un langage comme le Pascal, tandis que les tables LR canoniques ont plusieurs milliers d'états.

Avant de nous intéresser à la méthode SLR, à la base des deux autres méthodes, nous allons étudier les notions fondamentales et plus particulièrement la construction des ensembles d'items, étape préparatoire indispensable pour la construction des tables d'analyse.

4.3.7 Construction de la collection canonique

Notions de base

Nous définissons maintenant les principaux concepts nécessaires à la compréhension de l'algorithme SLR.

Définition 11 (Préfixes viables)
Les préfixes d'une proto-phrase droite qui peuvent apparaître sur la pile d'un analyseur par décalage-réduction sont appelés préfixes viables. Un préfixe

viable est un préfixe d'une proto-phrase droite qui ne s'étend pas au-delà de l'extrémité droite du manche le plus à droite de cette proto-phrase.

Définition 12 (Item LR(0))
Un item LR(0) (item en abrégé) d'une grammaire G est une production de G avec un point repérant une position de sa partie droite. La production $A \to \epsilon$ fournit uniquement l'item $A \to \cdot$.

Exemple 6 La production $A \to X\ Y\ Z$ fournit les quatre items :

$$\begin{aligned} A &\to \ \cdot X\ Y\ Z \\ A &\to X \cdot Y\ Z \\ A &\to X\ Y \cdot Z \\ A &\to X\ Y\ Z \cdot \end{aligned}$$

Définition 13 (Items valides)
Nous disons que l'item $A \to \beta_1 \cdot \beta_2$ est valide pour un préfixe viable $\alpha\beta_1$ s'il existe une dérivation :

$$S' \underset{d}{\overset{*}{\Rightarrow}} \alpha A w \underset{d}{\Rightarrow} \alpha\beta_1\beta_2 w$$

En général, un item est valide pour plusieurs préfixes viables.

Le fait que $A \to \beta_1 \cdot \beta_2$ soit valide pour un préfixe viable $\alpha\beta_1$ indique l'action, `décaler` ou `réduire`, à effectuer lorsque $\alpha\beta_1$ se trouve sur la pile. Si $\beta_2 \neq \epsilon$, il suggère que l'action doit être décaler. Si $\beta_2 = \epsilon$, $A \to \beta_1$ pouvant être le manche, l'action serait réduire par cette production.

Définition 14 (Collection LR(0) canonique)
Nous appelons collection LR(0) canonique une collection d'ensembles d'items LR(0).

Pour construire la collection LR(0) canonique pour une grammaire, nous définissons une grammaire augmentée et deux fonctions, Fermeture et Transition.

Définition 15 (Grammaire augmentée)
Si G est une grammaire d'axiome S, alors G', la grammaire augmentée de G, est G avec un nouvel axiome S' et une nouvelle production $S' \to S$.

Définition 16 (Fermeture(I))
Si I est un ensemble d'items pour une grammaire G, Fermeture(I) est l'ensemble d'items construit à partir de I par les deux règles :

1. *Initialement, placer chaque item de I dans Fermeture(I).*
2. *Si $A \to \alpha \cdot B\beta$ est dans Fermeture(I) et $B \to \gamma$ est une production, ajouter l'item $B \to \cdot \gamma$ à Fermeture(I), s'il ne s'y trouve pas déjà. Nous appliquons cette règle jusqu'à ce qu'aucun nouvel item ne puisse plus être ajouté à Fermeture(I).*

Définition 17 (Transition(I, X))
Soient I un ensemble d'items et X un symbole de la grammaire, nous appelons Transition(I, X) la fermeture de l'ensemble de tous les items $[A \rightarrow \alpha X \cdot \beta]$ obtenus en déplaçant d'un symbole vers la doite le point dans tous les items $[A \rightarrow \alpha \cdot X\beta]$ appartenant à I.

Exemple 7 Considérons la grammaire augmentée :

$$\begin{array}{ll} S' & \rightarrow S \\ S & \rightarrow SP\ S \\ S & \rightarrow SP\ V \\ SP & \rightarrow N\ P \end{array}$$

et $I = \{[S \rightarrow \cdot\ SP\ S]\}$.
Transition(I, SP) est :

$$\begin{array}{rcl} \mathsf{Transition}(I,\ SP) & = & \mathsf{Fermeture}(\{[S \rightarrow SP \cdot S]\}) \\ & = & \{[S \rightarrow SP \cdot S],\ [S \rightarrow \cdot\ SP\ S], \\ & & \ [S \rightarrow \cdot\ SP\ V],\ [SP \rightarrow \cdot\ N\ P]\} \end{array}$$

Algorithme de construction de la collection canonique

Données : G, grammaire augmentée, I_i, ensembles d'items.
Résultat : C, collection canonique des ensembles d'items LR(0) pour la grammaire augmentée G.

[1] $C := I_0 := \{\mathsf{Fermeture}([S' \rightarrow S])\}$
[2] $i := 1$ /* indice pour le nouvel ensemble d'item */
[3] répéter indéfiniment
  (a) répéter pour tous les ensembles d'items I_j
    (i) répéter pour tous les symboles de grammaire X
      1) $I_i := \{\mathsf{Transition}(I_j,\ X)\}$
      2) **Si** $I_i \neq \emptyset$ et $I_i \notin C$,
        alors $C := I_i$ et $i := i + 1$
    (ii) **Si** $j = i$,
      alors arrêter de répéter
  (b) retourner C

Exemple 8 Considérons la grammaire augmentée :

$$\begin{array}{ll} S' & \rightarrow S \\ S & \rightarrow SP\ S \\ S & \rightarrow SP\ V \\ SP & \rightarrow N\ P \end{array}$$

la collection canonique des ensembles d'items LR(0) pour cette grammaire est :

1. $I_0 = \{\mathsf{Fermeture}([S' \to \cdot\ S])\}$
   $= \{[S' \to \cdot\ S], [S \to \cdot\ SP\ S], [S \to \cdot\ SP\ V], [SP \to \cdot\ N\ P]\}$
2. $I_1 = \{\mathsf{Transition}(I_0, S)\}$
   $= \{\mathsf{Fermeture}([S' \to S \cdot\ ])\}$
   $= \{[S' \to S \cdot\ ]\}$
3. $I_2 = \{\mathsf{Transition}(I_0, SP)\}$
   $= \{\mathsf{Fermeture}([S \to SP \cdot S]), \mathsf{Fermeture}([S \to SP \cdot V])\}$
   $= \{[S \to SP \cdot S], [S \to SP \cdot V], [S \to \cdot\ SP\ S],$
   $[S \to \cdot\ SP\ V], [SP \to \cdot\ N\ P]\}$
4. $I_3 = \{\mathsf{Transition}(I_0, N)\}$
   $= \{\mathsf{Fermeture}([SP \to N \cdot P])\}$
   $= \{[SP \to N \cdot P]\}$
5. $I_4 = \{\mathsf{Transition}(I_2, V)\}$
   $= \{\mathsf{Fermeture}([S \to SP\ V \cdot\ ])\}$
   $= \{[S \to SP\ V \cdot\ ]\}$
6. $I_5 = \{\mathsf{Transition}(I_2, S)\}$
   $= \{\mathsf{Fermeture}([S \to SP\ S \cdot\ ])\}$
   $= \{[S \to SP\ S \cdot\ ]\}$
7. $I_6 = \{\mathsf{Transition}(I_2, SP)\}$
   sans création d'un nouvel ensemble car $I_6 = I_2$
8. $I_6 = \{\mathsf{Transition}(I_2, N)\}$
   sans création d'un nouvel ensemble car $I_6 = I_3$
9. $I_6 = \{\mathsf{Transition}(I_3, P)\}$
   $= \{\mathsf{Fermeture}([SP \to \cdot\ N\ P])\}$
   $= \{[SP \to N\ P \cdot\ ]\}$

4.3.8 Construction des tables d'analyse SLR

L'idée centrale de la méthode SLR est de construire tout d'abord, à partir de la grammaire, un automate fini déterministe pour reconnaître les préfixes viables. Les items sont regroupés en ensembles qui constituent les états de l'analyseur SLR. La collection LR(0) canonique fournit donc la base de construction des analyseurs SLR.

Automate fini déterministe représentant Transition

Exemple 9 La fonction Transition pour les ensembles d'items de la grammaire de l'exemple précédent, est donnée sous forme d'un diagramme de transition d'un automate fini déterministe D sur la figure 4.4 (voir page suivante).

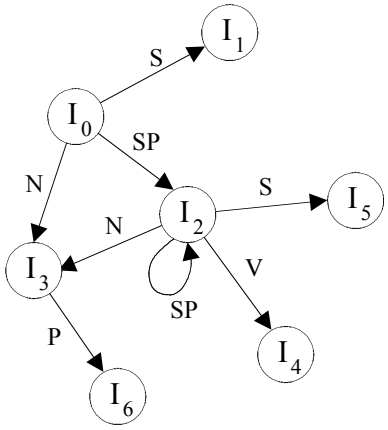


FIG. 4.4 – Automate de transition

Si chaque état de D de la figure 4.4 est un état final et I_0 est l'état initial, alors D reconnaît exactement les préfixes viables de la grammaire de l'exemple. Pour chaque grammaire, en fait, la fonction Transition de la collection canonique des ensembles d'items définit un automate fini déterministe (AFD en abrégé) qui reconnaît les préfixes viables de G.

De plus, avec cet AFD nous pouvons facilement calculer l'ensemble des items valides pour chaque préfixe viable qui peut apparaître sur la pile d'un analyseur LR. L'ensemble des items valides pour le préfixe viable γ est exactement l'ensemble des items atteint depuis l'état initial le long d'un chemin étiqueté γ dans l'AFD construit à partir de la collection canonique d'ensembles d'items dont les transitions sont données par Transition.

Avant de construire les tables d'analyse SLR, les fonctions Action et Successeur, regardons encore deux fonctions nécessaires, à savoir Premier(X) et Suivant(A).

Fonction Premier

Si α est une chaîne de symboles grammaticaux, Premier(α) désigne l'ensemble des terminaux qui commencent les chaînes qui se dérivent de α. Si $\alpha \stackrel{*}{\Rightarrow} \epsilon$, alors ϵ est aussi dans Premier(α).

Pour calculer Premier(X) pour tout symbole de la grammaire X, il faut appliquer les règles suivantes jusqu'à ce qu'aucun terminal ni ϵ ne puisse être ajouté aux ensembles de Premier.

1. Si X est un terminal, Premier(X) est $\{X\}$.
2. Si $X \rightarrow \epsilon$ est une production, ajouter ϵ à Premier(X).
3. Si X est un non-terminal et $X \rightarrow Y_1 Y_2 \ldots Y_k$ une production, mettre a dans Premier(X) s'il existe i tel que a est dans Premier(Y_i) et que ϵ est

dans tous les $\mathsf{Premier}(Y_1), \cdots, \mathsf{Premier}(Y_{i-1})$, c'est-à-dire $Y_1 \ldots Y_{i-1} \stackrel{*}{\Rightarrow} \epsilon$.
Si ϵ est dans $\mathsf{Premier}(Y_j)$ pour tous les $j = 1, 2, \ldots, k$, ajouter ϵ à $\mathsf{Premier}(X)$.
Si Y_1 ne se dérive pas en ϵ, ne rien ajouter à $\mathsf{Premier}(X)$, mais si $Y_1 \stackrel{*}{\Rightarrow} \epsilon$, ajouter $\mathsf{Premier}(Y_2)$, etc.

En d'autres termes, comme la figure 4.5 le montre, la fonction $\mathsf{Premier}(X)$ consiste à trouver l'ensemble des terminaux apparaissant dans les feuilles en développant les branches gauches à partir du nœud X.

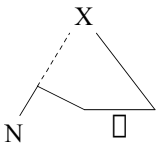


FIG. 4.5 – Éléments de l'ensemble $\mathsf{Premier}(X)$

Exemple 10 Considérons la grammaire :

1. $S \rightarrow SP\ S$
2. $S \rightarrow SP\ V$
3. $SP \rightarrow N\ P$

Par la règle 1,
$\mathsf{Premier}(V) = \{V\}$,
$\mathsf{Premier}(N) = \{N\}$,
$\mathsf{Premier}(P) = \{P\}$.
Par la règle 3,
$\mathsf{Premier}(SP) = \{N\}$,
$\mathsf{Premier}(S) = \mathsf{Premier}(SP) = \{N\}$.

Fonction Suivant

Pour chaque non-terminal A, $\mathsf{Suivant}(A)$ definit l'ensemble des terminaux a qui peuvent apparaître immédiatement à droite de A dans une proto-phrase, c'est-à-dire l'ensemble des terminaux a tels qu'il existe une dérivation de la forme $S \stackrel{*}{\Rightarrow} \alpha A a \beta$ où α et β sont des chaînes de symboles grammaticaux. Si A peut être le symbole le plus à droite d'une proto-phrase, alors \$ est dans $\mathsf{Suivant}(A)$.

Pour calculer $\mathsf{Suivant}(A)$ pour tous les non-terminaux A, il faut appliquer les règles suivantes jusqu'à ce qu'aucun terminal ne puisse être ajouté aux ensembles Suivant.

1. Mettre \$ dans $\mathsf{Suivant}(S)$, où S est l'axiome et \$ le marqueur droit indiquant la fin du texte source.
2. S'il y a une production $A \rightarrow \alpha B \beta$, le contenu de Premier($\beta$), excepté ϵ, est ajouté à $\mathsf{Suivant}(B)$.
3. S'il existe une production $A \rightarrow \alpha B$ ou une production $A \rightarrow \alpha B \beta$ telle que $\mathsf{Premier}(\beta)$ contient ϵ (c'est-à-dire $\beta \stackrel{*}{\Rightarrow} \epsilon$), les éléments de $\mathsf{Suivant}(A)$ sont ajoutés à $\mathsf{Suivant}(B)$.

Reformulons la règle 3. Dans le cas de $A \rightarrow \alpha B$, B apparaît au coin droit de l'arbre ayant comme racine A, et dans le cas de $A \rightarrow \alpha B \beta$, B apparaît juste à gauche du coin droit de l'arbre ϵ ayant comme racine A. La règle dit que dans ces cas, l'ensemble Suivant de la racine A peut être également l'ensemble Suivant de sa feuille droite B.

Exemple 11 Considérons la grammaire :

1. $S \rightarrow SP\ S$
2. $S \rightarrow SP\ V$
3. $SP \rightarrow N\ P$

Par la règle 1,
$\mathsf{Suivant}(S) = \{\$\}$.
Par la règle 2,
$\mathsf{Suivant}(SP) = \{\mathsf{Premier}(S), \mathsf{Premier}(V)\} = \{N, V\}$.

Algorithme de la méthode SLR

Nous pouvons à présent enfin construire les tables d'analyse SLR à partir de l'automate fini déterministe qui reconnaît les préfixes viables.

Données : G', grammaire augmentée.
Résultat : les tables d'analyse SLR des fonctions Action et Successeur pour la grammaire augmentée G'.

Méthode :

[1] Construire $C = \{I_0, I_1, \ldots, I_n\}$, la collection des ensembles d'items LR(0) pour G'.

[2] L'état i est construit à partir de I_i. Les actions d'analyse pour l'état i sont déterminées comme suit :
   (a) Si $[A \rightarrow \alpha \cdot a\beta]$ est dans I_i et $\mathsf{Transition}(I_i, a) = I_j$, remplir $\mathsf{Action}[i, a]$ avec « décaler et empiler l'état j ». Ici a doit être un terminal.
   (b) Si $[A \rightarrow \alpha \cdot]$ est dans I_i, remplir $\mathsf{Action}[i, a]$ avec « réduire par $A \rightarrow \alpha$ » pour tous les a de $\mathsf{Suivant}(A)$. Ici A ne doit pas être S'.
   (c) Si $[S' \rightarrow S \cdot]$ est dans I_i, remplir $\mathsf{Action}[i, \$]$ avec « accepter ».

Si les règles précédentes engendrent des actions conflictuelles, nous disons que la grammaire n'est pas SLR(1). Dans ce cas, l'algorithme échoue et ne produit pas d'analyseur.

[3] On construit les transitions Successeur pour l'état i pour tout non- terminal A en utilisant la règle : si Transition$(I_i, A) = I_j$, alors Successeur$[i, A] = j$.

[4] Toutes les entrées non définies par les règles [2] et [3] sont positionnées à « erreur ».

[5] L'état initial de l'analyseur est celui qui est construit à partir de l'ensemble d'items contenant $[S' \rightarrow \cdot S]$.

Les tables d'analyse formées des fonctions Action et Successeur déterminées par l'algorithme ci-dessus sont appelées tables SLR(1) pour G. On appelle un analyseur LR utilisant les tables SLR(1) pour G, analyseur SLR(1).

Exemple 12 Construisons les tables d'analyse SLR de la grammaire d'exemple pour laquelle nous avons déjà construit la collection canonique :

Fonction Action
État 0 :

1. Action$[0, N]$ = décaler 3.
   (Par l'item $[SP \rightarrow \cdot\ N\ P]$ et Transition$(I_0, N) = I_3$)

État 1 :

1. Action$[1, \$]$ = accepter.
   (Par l'item $[S' \rightarrow S\ \cdot\ ]$)

État 2 :

1. Action$[2, N]$ = décaler 3.
   (Par l'item $[SP \rightarrow \cdot\ N\ P]$ et Transition$(I_2, N) = I_3$)
2. Action$[2, V]$ = décaler 4.
   (Par l'item $[S \rightarrow SP \cdot V]$ et Transition$(I_2, V) = I_4$)

État 3 :

1. Action$[3, P]$ = décaler 6.
   (Par l'item $[SP \rightarrow N \cdot P]$ et Transition$(I_3, P) = I_6$)

État 4 :

1. Action$[4, \$]$ = réduire par $[S \rightarrow SP\ V]$.
   (Par l'item $[S \rightarrow SP\ V\ \cdot\ ]$ et Suivant$(S) = \{\$\}$)

État 5 :

1. Action$[5, N]$ = réduire par $[S \rightarrow SP\ S]$.
   (Par l'item $[SP \rightarrow SP\ S\ \cdot\ ]$ et Suivant$(S) = \{\$\}$)

État 6 :

1. $\mathsf{Action}[6, N] = \mathsf{Action}[6, V] =$ réduire par $[S \rightarrow SP\ V]$.
   (Par l'item $[SP \rightarrow N\ P \cdot\ ]$ et $\mathsf{Suivant}(SP) = \{N, V\}$)

Fonction Successeur
État 0 :

1. $\mathsf{Successeur}[0, S] = 1$
2. $\mathsf{Successeur}[0, SP] = 2$

État 2 :

1. $\mathsf{Successeur}[2, S] = 5$
2. $\mathsf{Successeur}[2, SP] = 2$

Nous obtenons alors les tables d'analyse SLR suivantes[6] :

État	Action				Successeur	
	N	P	V	\$	SP	S
0	d3				2	1
1				acc		
2	d3		d4		2	5
3		d6				
4				r2		
5				r1		
6	r3		r3			

4.3.9 Exemple d'analyse avec des tables SLR

Nous allons voir dans cette section un exemple d'analyse syntaxique avec les tables SLR que nous venons de construire. La grammaire est :

1. $S \rightarrow SP\ S$
2. $S \rightarrow SP\ V$
3. $SP \rightarrow N\ P$

La séquence d'entrée w est le résultat de l'analyse morphologique de la chaîne あかい　と　走る :

$$w = N\ P\ V$$

Initialement, l'analyseur LR est dans l'état 0 avec N comme premier symbole en entrée.

0　　N P V \$

L'action dans l'état 0 sur l'entrée N est d3, qui signifie décaler et empiler

[6] Où di signifie « décaler et empiler l'état i », ri signifie « réduire par la production i », acc signifie « accepter (fin d'analyse) » et une case vide « erreur ».

l'état 3.

0 N 3 P V $

L'action dans l'état 3 sur l'entrée P est d6, qui signifie décaler et empiler l'état 6.

0 N 3 P 6 V $

L'action dans l'état 6 sur l'entrée V est r3, qui signifie réduire par la production 3 : $SP \rightarrow N\ P$. Successeur pour l'état 0 sur SP est 2, donc SP et l'état 2 sont empilés.

0 SP 2 V $

L'action dans l'état 2 sur l'entrée V est d4, qui signifie décaler et empiler l'état 4.

0 SP 2 V 4 $

L'action dans l'état 4 sur l'entrée $ est r2, qui signifie réduire par la production 2 : $S \rightarrow SP\ V$. Successeur pour l'état 0 sur S est 1, donc S et l'état 1 sont empilés.

0 S 1 $

L'action dans l'état 1 sur l'entrée $ est accepter.
L'analyseur s'arrête, annonce la réussite finale de l'analyse et produit l'arbre syntaxique représenté figure 4.6.

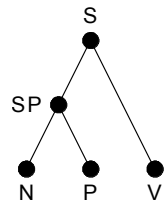


FIG. 4.6 – Arbre syntaxique résultat

4.4 Algorithme bi-directionnel

Nous allons maintenant nous intéresser à un autre type d'algorithme qui adopte à la fois les stratégies descendante et ascendante. L'analyse syntaxique dite de coin gauche est une méthode à la fois descendante et ascendante par nature.

Cette section est consacrée au mécanisme de cette analyse par la méthode du coin gauche. Nous présentons également une possibilité de réalisation d'un analyseur par la méthode du coin gauche, de type transducteur à pile. Avant d'entrer dans l'étude de l'algorithme, nous allons nous intéresser aux notions de base et présenter les automates à pile, connaissances nécessaires pour comprendre le principe des transducteurs à pile.

4.4.1 Coin gauche et Analyse par la méthode du coin gauche

Définissons tout d'abord les deux termes « coin gauche » et analyse par la méthode du coin gauche, et illustrons ensuite la définition par un exemple d'analyse.

Définition 18 (Coin gauche)
Le coin gauche d'une non ϵ-production est le symbole le plus à gauche (terminal ou non-terminal) de la partie droite.

Définition 19 (Analyse par la méthode du coin gauche)
Une analyse par la méthode du coin gauche d'une phrase est la séquence de productions utilisées pour créer les nœuds intérieurs d'un arbre syntaxique dans lequel tous les nœuds sont ordonnés comme suit :
Si le nœud n a p descendants directs $n_1, n_2, \ldots, n_p$, alors :
- *tous les nœuds du sous-arbre ayant comme racine n_1 précèdent n ;*
- *le nœud n précède tous les autres descendants, c'est à dire ses descendants directs $n_2, \ldots, n_p$ et tous leurs descendants.*

Les descendants de n_2 précèdent ceux de n_3, qui précèdent ceux de n_4, et ainsi de suite.

Dans une analyse par la méthode du coin gauche, le coin gauche d'une production est reconnu dans le sens ascendant et les autres symboles de la production sont reconnus dans le sens descendant.

Exemple 13 La figure 4.7 (voir page suivante) montre un arbre syntaxique pour la phrase *bbaaab* générée par la grammaire suivante :

1. $S \rightarrow AS$
2. $S \rightarrow BB$
3. $A \rightarrow bAA$
4. $A \rightarrow a$
5. $B \rightarrow b$
6. $B \rightarrow \epsilon$

Selon l'ordre des nœuds imposé par la définition de l'analyse par la méthode du coin gauche, le nœud n_2 et ses descendants précèdent n_1, qui est suivi par n_3 et ses descendants. Le nœud n_4 précède n_2, qui précède n_5, n_6 et leurs descendants. n_9 précède n_5, qui précède $n_1 0$, $n_1 1$ et leurs descendants. En continuant de cette façon, on obtient l'ordre des nœuds suivant :

$$n_4\, n_2\, n_9\, n_5\, n_{15}\, n_{10}\, n_{16}\, n_{11}\, n_{12}\, n_6\, n_1\, n_{13}\, n_7\, n_3\, n_{14}\, n_8$$

L'analyse par la méthode du coin gauche est la séquence de productions appliquées aux nœuds intérieurs dans cet arbre. L'analyse par la méthode du coin gauche pour *bbaaab* est donc 334441625.

Il est possible de concevoir un transducteur à pile qui fonctionne comme un analyseur par la méthode du coin gauche. Avant d'étudier un analyseur

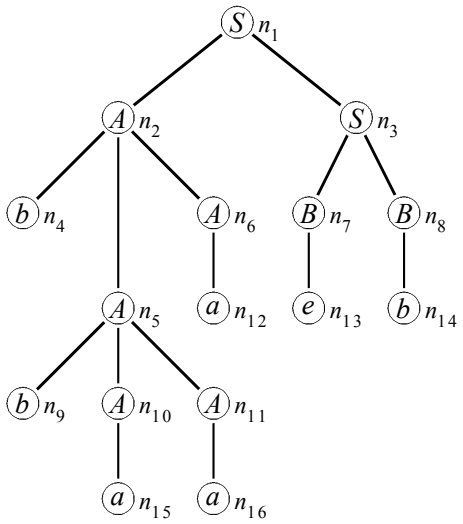


FIG. 4.7 – Arbre syntaxique

réalisé avec un transducteur à pile, nous allons nous intéresser aux connaissances supposées pré-acquises, à savoir les automates à pile et les transducteurs à pile.

4.4.2 Automate à pile

Un automate à pile (*Pushdown Automaton*, PDA en abrégé) est un accepteur (*recognizer*) qui constitue un modèle naturel pour les analyseurs syntaxiques des langages hors contexte. C'est un accepteur non-déterministe mono-directionnel, ayant une mémoire non bornée, une pile, c'est-à-dire un tableau potentiellement infini de cellules, géré par un protocole « dernier entré, premier sorti ». La figure 4.8 présente le schéma d'un automate à pile.

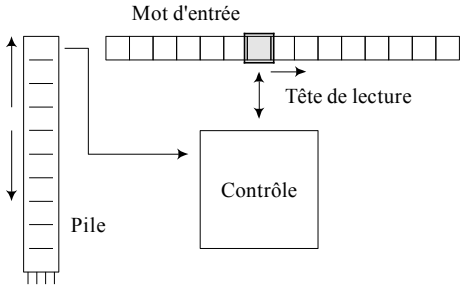


FIG. 4.8 – Automate à pile

Définition 20 (Automate à pile)
Un automate à pile est un septuple

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

où

- Q *est un ensemble fini de symboles d'état représentant les états possibles du contrôle ;*
- Σ *est un alphabet fini d'entrées ;*
- Γ *est un alphabet fini de symboles de liste à pile ;*
- δ *est une relation finie de transitions entre* $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ *et les sous-ensembles finis de* $Q \times \Gamma^*$ *;*
- $q_0 \in Q$ *est l'état initial du contrôle ;*
- $Z_0 \in \Gamma$ *est le symbole apparaissant initialement dans la liste à pile, appelé symbole de départ (strat symbole) ;*
- $F \subseteq Q$ *est l'ensemble des états finaux.*

Définition 21 (Configuration)
Une configuration de l'automate à pile P *est un triplet* (q, w, d) *dans* $Q \times \Sigma^* \times \Gamma^*$, *où*

- q *représente l'état courant du contrôle ;*
- w *représente la partie non utilisée de l'entrée. Le premier symbole de* w *est sous la tête de lecture. Si* $w = \epsilon$, *alors le mot d'entrée est entièrement lu ;*
- α *représente le contenu de la liste à pile. Le symbole le plus à gauche de* α *est au sommet de la pile. Si* $\alpha = \epsilon$, *alors la liste à pile est vide.*

Définition 22 (Transition)
Une transition par l'automate à pile P *est représentée par la relation binaire* $\underset{P}{\vdash}$ *(ou simplement* $\vdash$ *s'il n'y a pas d'ambiguïté possible sur l'automate à pile) entre les configurations. On la note :*

$$(q, aw, Z\alpha) \vdash (q', w, \gamma\alpha)$$

Si $a \neq \epsilon$, la configuration de la définition signifie que si P est dans une configuration telle que le contrôle est dans l'état q, que le symbole d'entrée courant est a, et que le symbole au sommet de la liste à pile est Z, alors P passe à une configuration dans laquelle le contrôle est dans l'état q', la tête de lecture est décalée à droite d'une case, et le symbole au sommet de la liste à pile est remplacé par la chaîne γ de symboles de la liste à pile.

Si $a = \epsilon$, alors la transition est appelée une ϵ-transition. Dans une ϵ-transition, le symbole d'entrée courant n'est pas pris en compte, et la tête de lecture n'est pas déplacée. Cependant, l'état du contrôle peut être changé, et le contenu de la mémoire peut être mis à jour. Il faut noter qu'une ϵ-transition peut apparaître même après la lecture entière du mot d'entrée.

Aucune transition n'est possible lorsque la liste à pile est vide.

On définit $\overset{i}{\vdash}$ pour $i \geq 0$, $\overset{*}{\vdash}$, et $\overset{+}{\vdash}$ de la façon habituelle. Donc, $\overset{*}{\vdash}$ et $\overset{+}{\vdash}$ sont respectivement la fermeture réflexive-transitive et la fermeture transitive de $\vdash$.

Une configuration initiale de P a une forme (q_0, w, Z_0) pour $w \in \Sigma^*$. Le contrôle est dans l'état initial, l'entrée contient la chaîne à reconnaître, et la liste à pile ne contient que le symbole Z_0. Une configuration finale a une forme (q, ϵ, α), où q est dans F et α dans Γ^*.

On dit qu'une chaîne w est acceptée par P si $(q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \alpha)$ pour $q \in F$ et $\alpha \in \Gamma^*$. Le langage défini par P, noté $L(P)$, est l'ensemble des chaînes acceptées par P. $L(P)$ est appelé langage de l'automate à pile.

4.4.3 Transducteur à pile

Un transducteur à pile (*Pushdown Transducer*, PDT en abrégé) est obtenu par un automate à pile avec sortie. À chaque transition, l'automate émet une chaîne de sortie de longueur finie.

Définition 23 (Transducteur à pile)
Un transducteur à pile P est un octuple, $(Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, F)$, où tous les symboles ont le même sens que pour un PDA, excepté :

- *Δ est un alphabet fini de symboles de sortie ;*
- *δ est une relation finie de transitions entre $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ et les sous-ensembles finis de $Q \times \Gamma^* \times \Delta^*$.*

Définition 24 (Configuration)
Nous définissons une configuration du transducteur à pile P comme un quadruplet (q, x, α, y), où q, x et α sont les mêmes que pour un PDA et y est la chaîne de sortie émise à cette transition.

Si $\delta(q, a, Z)$ contient (r, α, z), alors on note :

$$(q, ax, Z\gamma, y) \vdash (r, x, \alpha\gamma, yz)$$

pour tout $x \in \Sigma^*$, $\gamma \in \Gamma^*$ et $y \in \Delta^*$.

On dit que y est une sortie pour x si $(q_0, x, Z_0, \epsilon) \overset{*}{\vdash} (q, \epsilon, \alpha, y)$ pour $q \in F$ et $\alpha \in \Gamma^*$. La transition définie par P, notée $\tau(P)$, est :

$$\{(x, y) | (q_0, x, Z_0, \epsilon) \overset{*}{\vdash} (q, \epsilon, \alpha, y) \text{ pour } q \in F \text{ et } \alpha \in \Gamma^*\}$$

4.4.4 Analyseur par la méthode du coin gauche par transducteur à pile

Comme il a été dit plus haut, il est possible de concevoir un analyseur par la méthode du coin gauche avec un transducteur à pile.

À partir d'une grammaire hors contexte $G = (N, \Sigma, P, S)$, on peut construire un PDT $M = (\{q\},\ \Sigma,\ N \times N \cup N \cup \Sigma,\ \Delta,\ \delta,\ q,\ S,\ \emptyset)$, qui est un analyseur par la méthode du coin gauche pour G.

L'analyseur utilise comme symboles de liste à pile, des non-terminaux, des terminaux et des symboles spéciaux de la forme $[A, B]$, où A et B sont des non-terminaux. Les non-terminaux et les terminaux apparaissant dans la liste à pile sont des buts à reconnaître dans un sens descendant. Dans un symbole $[A, B]$, A est le but courant à reconnaître et B le non-terminal qui vient d'être reconnu de manière ascendante.

L'alphabet de sortie Δ est l'ensemble des numéros de production $\{1,\ 2,\ \ldots,\ p\}$, et δ est défini comme suit :

1. Supposons que $A \to \alpha$ soit la $i^{\text{ème}}$ production dans P.
   (a) Si α est de la forme $B\beta$, où $B \in N$, alors $\delta(q, \epsilon, [C, B])$ contient $(q, \beta[C, A], i)$ pour tout $C \in N$. Ce qui signifie que l'on a reconnu le coin gauche B de manière ascendante, et que l'on considère donc les symboles dans β comme un nouveau but à reconnaître de manière descendante. Une fois β reconnu, A l'est également.
   (b) Si α ne commence pas par un non-terminal, alors $\delta(q, \epsilon, C)$ contient $(q, \alpha[C, A], i)$ pour tout non-terminal C. Ce qui signifie que la reconnaissance de α entraîne celle du non-terminal A.
2. $\delta(q, \epsilon, [A, A])$ contient (q, ϵ, ϵ) pour tout $A \in N$. Ce qui signifie qu'une instance du but A que l'on cherchait a été reconnue. Si cette instance de A n'est pas un coin gauche, on supprime $[A, A]$ de la liste à pile, ce qui signifie que cette instance de A était le but recherché.
3. $\delta(q, a, a) = \{(q, \epsilon, \epsilon)\}$ pour tout $a \in \Sigma$. Ce qui signifie que le but courant est un terminal correspondant au symbole d'entrée courant. Le but étant satisfait, il est supprimé.

M définit la traduction $\{(w, \pi) | w \in L(G)$ et π est un analyseur par la méthode du coin gauche pour $w\}$.

Exemple 14 Considérons la grammaire hors contexte $G = (N, \Sigma, P, S)$ avec les productions :

1. $E \to E + T$
2. $E \to T$
3. $T \to F \uparrow T$
4. $T \to F$
5. $F \to (E)$
6. $F \to a$

Un analyseur non déterministe par la méthode du coin gauche pour G est le PDT :

$$M = (\{q\}, \Sigma, N \times N \cup N \cup \Sigma, \{1, 2, \ldots, 6\}, \delta, q, E, \emptyset)$$

où δ est défini comme suit pour tout $A \in N$ selon les règles décrites précédemment :

1. (a) i. $\delta(q, \epsilon, [A, E])$ contient $(q, +T[A, E], 1)$;
   ii. $\delta(q, \epsilon, [A, T])$ contient $(q, [A, E], 2)$;
   iii. $\delta(q, \epsilon, [A, F])$ contient $(q, \uparrow T[A, T], 3)$ et $(q, [A, T], 4)$;
   (b) $\delta(q, \epsilon, A)$ contient $(q, (E)[A, F], 5)$ et $(q, a[A, F], 6)$;
2. $\delta(q, \epsilon, [A, A])$ contient (q, ϵ, ϵ) ;
3. $\delta(q, a, a) = \{(q, \epsilon, \epsilon)\}$ pour tout $a \in \Sigma$.

Essayons maintenant d'analyser la phrase d'entrée $a \uparrow a + a$ en utilisant M. Comme le PDT n'a qu'un état, nous l'ignorons.

1. La configuration initiale est
$$(a \uparrow a + a, E, \epsilon)$$
2. La seconde règle de 1(b) étant applicable, le PDT passe à la configuration
$$(a \uparrow a + a, a[E, F], 6)$$
3. Le coin gauche a a été généré en utilisant la production 6. Le symbole a est alors comparé avec le symbole d'entrée courant
$$(\uparrow a + a, [E, F], 6)$$
4. On peut maintenant utiliser la première règle dans 1(a)iii. pour obtenir
$$(\uparrow a + a, \uparrow T[E, T], 63)$$
5. Le coin gauche de la production $T \rightarrow F \uparrow T$ sera reconnu une fois que l'on aura trouvé $\uparrow$ et T. On peut alors entrer dans les configurations suivantes :
$$\begin{aligned} (a + a, T[E, T], 63) &\vdash (a + a, a[T, F][E, T], 636) \\ &\vdash (+a, [T, F][E, T], 636) \\ &\vdash (+a, [T, T][E, T], 6364) \end{aligned}$$
6. En continuant, on peut terminer avec la séquence suivante de configurations :
$$\begin{aligned} (+a, [E, E], 63642) &\vdash (a, a[T, F][E, E], 6364216) \\ &\vdash (\epsilon, [T, F][E, E], 6364216) \\ &\vdash (\epsilon, [T, T][E, E], 63642164) \\ &\vdash (\epsilon, \epsilon, 63642164) \end{aligned}$$

L'arbre d'analyse est présenté sur la figure 4.9 (voir page suivante). Les flèches montrant le sens des reconnaissances, on peut constater la nature bi-directionnelle de l'analyse : le coin gauche d'une production est reconnu dans un sens ascendant et les autres symboles de la production sont reconnus dans un sens descendant.

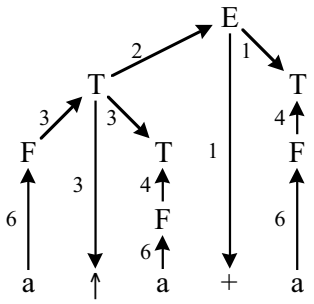


FIG. 4.9 – Analyse par la méthode du coin gauche

Chapitre 5

Analyseurs syntaxiques japonais

5.1 Introduction

Ce chapitre est consacré à l'étude des analyseurs syntaxiques réalisés au Japon. Nous présentons trois systèmes d'analyse syntaxique : SAX[1], MSLR[2] et KNP[3].

Le premier critère du choix des systèmes à étudier a été la disponibilité de la totalité des codes sources sur Internet. Le deuxième point pris en compte lors du choix était la diversité des méthodes adoptées par chaque système, permettant de mieux contraster leurs avantages et leurs inconvénients.

Nous allons étudier tout d'abord le système SAX, et nous nous intéressons ensuite au système MSLR, puis au système KNP. Enfin, la dernière partie sera consacrée à l'étude des résultats d'analyse obtenus par chacun des systèmes et à l'analyse de leurs points forts et de leurs erreurs.

5.2 SAX

5.2.1 Introduction

Le système SAX est un analyseur syntaxique développé dans un laboratoire du *Nara Institute of Science and Technology* (NAIST). Ce système compile une grammaire écrite en DCG (*Definite Clause Grammars*) à partir de laquelle il génère un programme d'analyse syntaxique écrit en langage PROLOG.

La DCG est une grammaire permettant de créer facilement un système d'analyse syntaxique de type descendante en profitant du mécanisme du

[1] `http://chasen.aist-nara.ac.jp/sax.html`

[2] `http://tanaka-www.cs.titech.ac.jp/pub/mslr/index-j.html`

[3] `http://www-lab25.kuee.kyoto-u.ac.jp/nl-resource/knp.htm`

langage PROLOG. Cependant un système créé de cette manière présente en général un grand défaut dû au problème de la récursivité à gauche. Le système SAX a comme particularité d'avoir résolu ce problème et d'avoir amélioré l'efficacité pour les grammaires de taille importante, en générant des programmes PROLOG de type ascendant sans retour en arrière, qui exécutent une analyse similaire à celle réalisée avec un algorithme de *Bottom-up Chart Parsing*.

L'étude sur SAX est constituée principalement de la présentation du principe de fonctionnement. Nous commencerons par l'architecture du système et nous nous intéresserons à la représentation des arcs dans le système. Nous étudierons ensuite la méthode de transformation de la grammaire DCG en programme PROLOG, le traitement des arcs installés et le traitement de fin d'analyse. Ces études s'appuie essentiellement sur le mode d'emploi du système distribué avec les codes sources. Certains détails diffèrent cependant de l'implantation réelle. Nous conserverons toutefois l'explication du manuel telle quelle, afin de mieux comprendre non pas l'implantation mais le principe de fonctionnement.

Enfin, nous compléterons cette étude du principe de fonctionnement du système par la présentation d'une possibilité d'amélioration algorithmique en nous appuyant sur l'article [23] publié par les créateurs du système SAX.

5.2.2 Architecture du système

Nous étudions dans cette section l'architecture du système.

SAX est un système qui transforme une grammaire écrite en DCG (*Definite Clause Grammars*) en un programme PROLOG d'analyse syntaxique à base de *Bottom-up Chart Parsing*, et qui réalise une analyse syntaxique de la phrase entrée.

L'architecture du système est représentée par le schéma 5.1 (voir page suivante). Le système est principalement constitué des deux parties suivantes :

- Module « sax_trans » : ce module transforme la grammaire en un programme d'analyse syntaxique (**Clauses SAX**).
- Module « sax » : ce module transforme la phrase entrée en but PROLOG (**But SAX**) et exécute le programme d'analyse syntaxique réalisé en clauses SAX.

5.2.3 Représentation des arcs dans le présent système

Rappelons d'abord le mécanisme de *Bottom-up Chart Parsing* qui est à la base de ce système.

Les opérations principales de cet algorithme sont :

a) **Initialisation**

La présence d'un arc non actif étiqueté A, à la position [t, u], avec la grammaire H :- B1, B2, ..., Bn. (où B1 = A), entraîne l'installation d'un

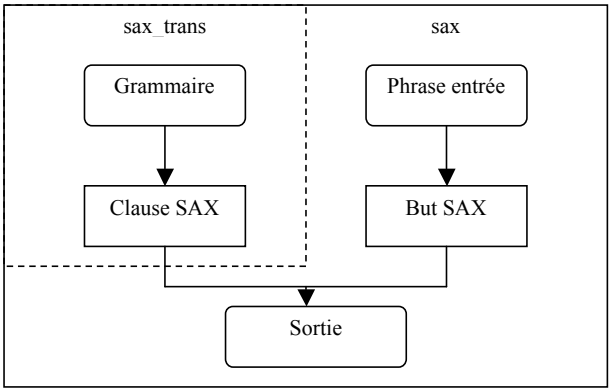
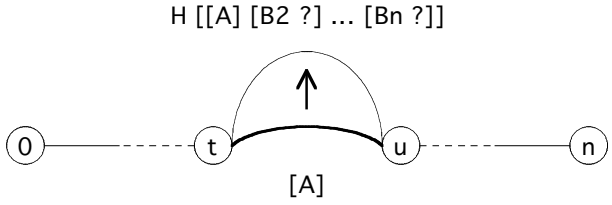
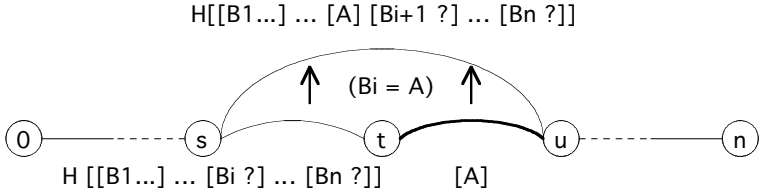


FIG. 5.1 – Architecture du système SAX

arc actif à la position [t, u], étiqueté H[[A] [B2 ?] ... [Bn ?]]. Le schéma suivant montre ce principe d'initialisation.



b) **Complétion**
La présence à la position [t, u] d'un arc non actif étiqueté A, et à la position [s, t] d'un arc actif étiqueté H[[B1...]... [Bi ?] ... [Bn ?]], et l'égalité Bi = A, entraînent l'installation d'un arc actif (ou d'un arc non actif si i = n) à la position [s, u] étiqueté H[[B1...]... [A] [Bi+1 ?] ... [Bn ?]]. Le schéma suivant montre ce principe de complétion.



Étudions maintenant comment le système représente les arcs actif et non actif avec un exemple concret afin de mieux comprendre.

Soit la grammaire :

$$sv \rightarrow v\ sn\ pp$$

La présence de l'arc non actif [v] entraîne par initialisation la création d'un arc actif étiqueté [sv [v] [sn ?] [pp ?]]. Nous obtenons ensuite, par complétion avec l'arc non actif [sn], un arc actif étiqueté [sv [v] [sn] [pp ?]]. Ces arcs actifs représentent jusqu'à quel symbole non terminal de la partie droite de la grammaire la structure est satisfaite. La méthode du système SAX consiste à représenter cette information en insérant des identificateurs uniques entre chaque symbole non terminal. Ces identificateurs représentent donc l'ensemble des informations que portait l'étiquette d'un arc actif. Par exemple, par insertion des identificateurs suivants :

$$\text{sv} \rightarrow \text{v id1 sn id2 pp}$$

les deux arcs actifs décrits précédemment peuvent être étiquetés respectivement *id1* et *id2*.

Les arcs, actif et non actif, sont représentés dans le système comme suit :

Arc non actif : *ie(NTerm, S)*
NTerm, symbole non terminal, est l'étiquette de l'arc correspondant et S est l'information de position comprenant la liste des **arcs actifs précédents**.

Arc actif : *ae(Id, S)*
Id est un identificateur correspondant à l'étiquette de l'arc actif et S est l'information de position comprenant la liste des **arcs actifs précédents**.

Les **arcs actifs précédents** sont les arcs qui précédent l'arc en cours de traitement et qui lui sont directement connectés. Les arcs actifs précédents d'un arc partant de la position t, sont des arcs se terminant à la position t. En effet, pour réaliser la complétion avec un arc non actif A, il faut chercher, parmi les arcs existants, des arcs actifs permettant la réalisation d'une complétion par A. Il suffit pour cela d'examiner uniquement les arcs actifs le précédant directement. Ainsi, tous les arcs portent comme information de position leurs arcs actifs précédents.

5.2.4 Transformation de la grammaire DCG en programme prolog (Clauses SAX)

Regardons maintenant comment le système transforme la grammaire DCG en clauses SAX. Illustrons l'explication avec la même grammaire que dans l'exemple précédent :

$$\text{sv} \rightarrow \text{v id1 sn id2 pp}$$

Cette règle peut être utilisée par les trois cas de complétion et d'initialisation décrits ci-dessous.

- Avec l'arc non actif `ie(v, St)` situé en $[t, u]$, installation de l'arc actif `ae(id1, St)` en $[t, u]$ par initialisation.
- S'il y a, dans la liste des arcs précédents l'arc non actif `ie(sn, St)` situé en $[t, u]$, l'arc actif `ae(id1, Ss)` situé en $[s, t]$, alors installation de l'arc actif `ae(id2, Ss)` en $[s, u]$ par complétion.
- S'il y a, dans la liste des arcs précédents l'arc non actif `ie(pp, St)` situé en $[t, u]$, l'arc actif `ae(id2, Ss)` situé en $[s, t]$, alors installation de l'arc non actif `ie(sv, Ss)` en $[s, u]$ par complétion.

Autrement dit, à partir d'une règle de grammaire DCG quelconque possédant des identificateurs, telle que :

$$\text{a} \rightarrow \text{b, id1, c, id2, d}$$

peuvent être créées trois procédures PROLOG, dénommées clauses SAX, comme suit :

1. Clause SAX appelée lors de l'installation de l'arc non actif de coin gauche b :
   ajout de l'arc actif étiqueté id1 dans la liste des arcs se terminant sur le même nœud que b.
2. Clause SAX appelée lors de l'installation de l'arc non actif de c :
   si et seulement si il existe, dans la liste des arcs actifs précédents, l'arc actif étiqueté id1, alors ajout de l'arc actif étiqueté id2 dans la liste des arcs se terminant sur le même nœud que c.
3. Clause SAX appelée lors de l'installation de l'arc non actif de d :
   si et seulement si il existe, dans la liste des arcs actifs précédents, l'arc actif étiqueté id2, alors installation de l'arc non actif de a dans la liste des arcs se terminant sur le même nœud que d.

Ainsi, on obtient à partir de notre règle de grammaire les clauses SAX suivantes :

```
v(St, Eu, EuTail) :-                              %1
  Eu = [ae(id1, St)|EuTail].                      %1-2

sn([], Eu, Eu) :- !.                              %2-1
sn([ae(id1, Ss)|StTail], Eu, EuTail) :-           %2-2
  Eu = [ae(id2, Ss)|Eu1],                         %2-2-1
  sn(StTail, Eu1, EuTail).                        %2-2-2
sn([_|StTail], Eu, EuTail) :-                     %2-3
  sn(StTail, Eu, EuTail).                         %2-3-1

pp([], Eu, Eu) :- !.                              %3-1
pp([ae(id2, Ss)|StTail], Eu, EuTail) :-           %3-2
  Eu = [ie(sv, Ss)|Eu1],                          %3-2-1
  pp(StTail, Eu1, EuTail).                        %3-2-2
pp([_|StTail], Eu, EuTail) :-                     %3-3
  pp(StTail, Eu, EuTail).                         %3-3-1
```

Le premier argument *St* de chaque prédicat *NTerm(St, Eu, EuTail)* est la liste des arcs actifs précédents de l'arc non actif *ie(NTerm, St)*. Le troisième *EuTail* est la liste reçue des arcs se terminant au même nœud que l'arc non actif *ie(NTerm, St)*. Le deuxième argument *Eu* est la liste complétée par l'ajout à la liste *EuTail* des nouveaux arcs créés. Regardons de plus près chaque clause.

- Clause 1 : Clause SAX du cas 1.
  - Clause 1-1 : ajout à la liste de l'arc actif ae(id1, St) créé par initialisation. Ce nouvel arc prend comme liste des arcs actifs précédents la même que celle de l'arc non actif ie(v, St).
- Clause 2 : Clause SAX du cas 2.
  - Clause 2-1 : clause d'arrêt.
  - Clause 2-2 : vérification de la liste des arcs actifs précédents. L'arc traité (c'est-à-dire la tête de liste) est l'arc actif ae(id1, Ss).
    - Clause 2-2-1 : ajout à la liste de l'arc actif ae(id2, Ss) créé par complétion. Ce nouvel arc prend comme liste des arcs actifs précédents la même que celle de l'arc actif ae(id1, Ss).
    - Clause 2-2-2 : passage au traitement du prochain arc à traiter.
  - Clause 2-3 : vérification de la liste des arcs actifs précédents. L'arc traité (c'est-à-dire la tête de liste) n'est pas l'arc actif ae(id1, Ss).
    - Clause 2-3-1 : passage au traitement du prochain arc à traiter.
- Clause 3 : Clause SAX du cas 3.
  - Clause 3-1 : clause d'arrêt.
  - Clause 3-2 : vérification de la liste des arcs actifs précédents. L'arc traité (c'est-à-dire la tête de la liste) est l'arc actif ae(id2, Ss).
    - Clause 3-2-1 : ajout à la liste de l'arc non actif ie(sv, Ss) créé par complétion. Ce nouvel arc prend comme liste des arcs actifs

précédents la même que celle de l'arc actif ae(id2, Ss).
  - Clause 3-2-2 : passage au traitement du prochain arc à traiter.
- Clause 3-3 : vérification de la liste des arcs actifs précédents. L'arc traité (c'est-à-dire la tête de liste) n'est pas l'arc actif ae(id2, Ss).
  - Clause 3-3-1 : passage au traitement du prochain arc à traiter.

Nous étudions maintenant comment traiter un arc se terminant à une position donnée.

5.2.5 Représentation du but

Le système lit les phrases et les analyse une par une. Il considère le caractère de retour à la ligne comme une fin de phrase. D'autres caractères de ponctuation tels que le point, le point d'interrogation ou la virgule, sont considérés comme une lettre. Le but d'analyse est créé à partir d'une phrase lue $[Mot_1, Mot_2, Mot_3, \ldots, Mot_m]$ en la transformant en la clause suivante, appelée But SAX :
`process(E1, S1), process(E2, S2), ..., process(Em, Sm), fin(Sm).`
où :

- E_j : est la liste des arcs non actifs *ie(NTerm, Sj-1)* créés à partir de Mot_j, constitués d'un symbole non terminal *NTerm* et d'une variable qui recevra la liste de ses arcs actifs précédents S_{j-1}.
- S_j : est la variable qui recevra la liste des arcs actifs se terminant au même nœud que le nœud où les arcs appartenant à la liste E_j se terminent.

E_j peut comprendre plusieurs arcs non actifs, car Mot_j peut être une polysémie. La liste des arcs actifs précédents *S0* de *Nterm1* contient un seul arc actif étiqueté par l'identificateur particulier *begin* comme *ae(begin, t)*.
Par exemple, à partir de la phrase lue [pierre, aime, martine], est créé un but SAX tel que :

```
process([ie(sn, [ae(begin, t)])], S1),
process([ie(v, S1)], S2),
process([ie(sn, S2)], S3),
fin(S3).
```

Nous allons regarder dans la section suivante de quelle manière, avec ce but, l'analyse se réalise.

5.2.6 Traitement d'un arc se terminant à une position donnée

La procédure d'analyse consiste à ajouter un nouvel arc à la liste rassemblant les arcs se terminant en un même nœud, chaque fois que la complétion ou l'initialisation se réalise. Ainsi, la procédure *process* doit être constituée de la façon suivante :

```
process([], []) :- !.                                    %1
process([ie(NTerm, St)|EuTail], Su) :-                   %2
  procedure(NTerm, St, NewEu, EuTail),                   %2-1
  process(NewEu, Su).                                    %2-2
process([Active|EuTail], [Active|SuTail]) :-             %3
  process(EuTail, SuTail).                               %3-1
```

Regardons de plus près chaque clause.

- Clause 1 : clause d'arrêt
- Clause 2 : si l'arc à traiter (c'est-à-dire la tête de liste du premier argument) est un arc non actif, alors on exécute le corps de la clause.
  - Clause 2-1 : consultation de la grammaire : exécution du prédicat `procedure(NTerm, St, NewEu, EuTail)`. Ce prédicat réalise la complétion et l'initialisation pour l'arc non actif *ie(NTerm, St)* et ajoute à la liste *EuTail* les nouveaux arcs résultants, créant ainsi la nouvelle liste *NewEu*. C'est-à-dire, qu'il appelle le prédicat que nous avons vu dans la section précédente : `NTerm(St, NewEu, EuTail)`.
  - Clause 2-2 : passage au traitement du prochain arc à traiter.
- Clause 3 : si l'arc à traiter est un arc actif, on le met dans la liste des arcs actifs précédents *Su*.

Le but SAX est terminé par le prédicat *fin(Sm)*. Ce prédicat exécute des opérations telles que l'affichage du résultat en cas d'analyse réussie. Nous allons, à présent, étudier comment connaître le résultat d'analyse, succès ou échec.

5.2.7 Fin d'analyse

Avant la transformation de la grammaire, doit être défini le symbole de début qui est le symbole non terminal représentant la racine de toute grammaire (symbole initial). Supposons que nous ayons défini le symbole de début par *STerm*. La réussite de l'analyse entraîne l'installation d'un arc non actif étiqueté par *STerm*, et ce entre la tête et la fin de phrase. Le mécanisme de signalisation de réussite d'analyse du système consiste alors en les deux étapes suivantes :

1) Si un arc non actif étiqueté par *STerm* est installé et que dans la liste de ses arcs actifs précédents se trouve l'arc actif particulier *ae(begin, t)*, alors installation d'un arc actif particulier *ae(idend, t)*.
2) Si dans la liste finale *Sm*, qui est l'argument du prédicat *fin*, se trouve un arc actif particulier *ae(idend, t)*, alors l'analyse est réussie et on a obtenu au moins un résultat.

En effet, comme l'arc actif particulier *ae(begin, t)* se trouve uniquement dans la liste *S0*, la présence d'un arc actif particulier *ae(idend, t)* dans la liste finale *Sm*, signifie l'installation de l'arc non actif étiqueté par *STerm* entre

la position [0, m], c'est-à-dire entre le début et la fin de phrase. Ces deux opérations se réalisent avec les procédures Prolog suivantes :
Opération 1 :

```
STerm([], Eu, Eu) :- !.                              %1
STerm ([ae(begin, t)|StTail], Eu, EuTail) :-         %2
  Eu = [ae(idend, t)|Eu1],                           %2-1
  STerm (StTail, Eu1, EuTail).                       %2-2
STerm ([_|StTail], Eu, EuTail) :-                    %3
  STerm (StTail, Eu, EuTail).                        %3-1
```

Opération 2 :

```
fin([]) :- !.                                        %1
fin ([ae(idend, t)|SmTail]) :-                       %2
  procedure_pour_le_résultat,                        %2-1
  fin (SmTail).                                      %2-2
fin ([_|SmTail]) :-                                  %3
  fin (SmTail).                                      %3-1
```

Regardons de plus près chaque clause.
Opération 1 :

- Clause 1 : clause d'arrêt.
- Clause 2 : vérification de la liste des arcs actifs précédents. L'arc traité (c'est-à-dire la tête de la liste) est l'arc actif `ae(begin, t)`.
  - Clause 2-1 : ajout à la liste de l'arc actif `ae(idend, t)`.
  - Clause 2-2 : passage au traitement du prochain arc à traiter.
- Clause 3 : vérification de la liste des arcs actifs précédents. L'arc traité (c'est-à-dire la tête de liste) n'est pas l'arc actif `ae(idend, t)`.
  - Clause 3-1 : passage au traitement du prochain arc à traiter.

Opération 2 :

- Clause 1 : clause d'arrêt
- Clause 2 : si l'arc à traiter (c'est-à-dire la tête de liste du premier argument) est l'arc actif `ae(idend, t)`, alors exécution du corps de la clause.
  - Clause 2-1 : exécution du prédicat `procedure_pour_résultat`. Ce prédicat réalise des opérations concernant la réponse obtenue telles que l'affichage du résultat.
  - Clause 2-2 : passage au traitement du prochain arc à traiter.
- Clause 3 : si l'arc à traiter n'est pas un arc actif `ae(idend, t)`, alors passage au traitement du prochain arc à traiter.

5.2.8 Exemple d'exécution de l'analyse d'une phrase

Considérons la grammaire suivante :

$S \rightarrow SP\ SV$
$SP \rightarrow Sub\ P$
$SP \rightarrow Q\ P$
$SV \rightarrow V$

Selon l'algorithme que nous venons d'étudier, nous obtenons à partir de cette grammaire les clauses SAX suivantes.

```
%%%%%%%%%%%%%%%%%%%
%% grammaire
%%%%%%%%%%%%%%%%%%%
```

```
% s -> sp, id1, sv.

sp(St, Eu, EuTail) :-                    % (a)
  Eu = [ae(id1, St)|EuTail].             % (a) - 1
sv([], Eu, Eu) :-!.                      % (b)
sv([ae(id1, Ss)|StTail], Eu, EuTail) :-  % (c)
  Eu = [ie(s, Ss)|Eu1],                  % (c) - 1
  sv(StTail, Eu1, EuTail).               % (c) - 2
sv([_|StTail], Eu, EuTail) :-            % (d)
  sv(StTail, Eu, EuTail).                % (d) - 1

% sp -> sub, id2, p.
% sp -> q, id3, p.

sub(St, Eu, EuTail) :-                   % (e)
  Eu = [ae(id2, St)|EuTail].             % (e) - 1
q(St, Eu, EuTail) :-                     % (f)
  Eu = [ae(id3, St)|EuTail].             % (f) - 1
p([], Eu, Eu) :-!.                       % (g)
p([ae(id2, Ss)|StTail], Eu, EuTail) :-   % (h)
  Eu = [ie(sp, Ss)|Eu1],                 % (h) - 1
  p(StTail, Eu1, EuTail).                % (h) - 2
p([ae(id3, Ss)|StTail], Eu, EuTail) :-   % (i)
  Eu = [ie(sp, Ss)|Eu1],                 % (i) - 1
  p(StTail, Eu1, EuTail).                % (i) - 2
p([_|StTail], Eu, EuTail) :-             % (j)
  p(StTail, Eu, EuTail).                 % (j) - 1
```

```
% sv -> v.

v(St, Eu, EuTail) :-          % (k)
  Eu = [ie(sv, St)|EuTail].   % (k) - 1

% symbole initial

s([], Eu, Eu) :-!.                          % (l)
s([ae(begin, t)|StTail], Eu, EuTail) :-     % (m)
  Eu = [ae(idend, t)|Eu1],                  % (m) - 1
  s(StTail, Eu1, EuTail).                   % (m) - 2
s([_|StTail], Eu, EuTail) :-                % (n)
  s(StTail, Eu, EuTail).                    % (n) - 1
```

Nous utilisons les procédures `process` et `fin` que nous avons étudiées dans cette section.

```
%%%%%%%%%%%%%%%%%%%
%% process
%%%%%%%%%%%%%%%%%%%

process([], []) :-!.                              % 1
process([ie(NTerm, St)|EuTail], Su) :-            % 2
  procedure(NTerm, St, NewEu, EuTail),            % 2 - 1
  process(NewEu, Su).                             % 2 - 2
process([Active|EuTail], [Active|SuTail]) :-      % 3
  process(EuTail, SuTail).                        % 3 - 1

%%%%%%%%%%%%%%%%%%%
%% fin
%%%%%%%%%%%%%%%%%%%

fin([]) :-!.                      % 4
fin([ae(idend, t)|SmTail]) :-     % 5
  procedure_fin,                  % 5 - 1
  fin(SmTail).                    % 5 - 2
fin([_|SmTail]) :-                % 6
  fin(SmTail).                    % 6 - 1
```

Réalisons une analyse de la phrase :

あかい　と　走る.

La séquence d'entrée w est le résultat de l'analyse morphologique de la phrase à analyser :

$$w = Sub\ P\ V$$

Le but SAX créé à partir de w est :

```
process([ie(sub,[ae(begin,t)]),  % 1
  ie(q, [ae(begin,t)])], S1),    % 1'
process([ie(p,S1)],S2),          % 2
process([ie(v, S2)], S3),        % 3
fin(S3).                         % 4
```

La requête avec ce but SAX se déroule comme suit :

```
% Appel du terme 1 du but
? process([ie(sub,[ae(begin,t)]), ie(q, [ae(begin,t)])], S1)
  % Unification avec la tête de la clause 2.
  NTerm = sub
  St = [ae(begin,t)]
  EuTail = [ie(q, [ae(begin,t)])]
  S1 = Su = ?

  % Appel du premier terme du corps  2-1
  ? procedure(sub,[ae(begin,t)], NewEu, [ie(q, [ae(begin,t)])])
   % Appel du prédicat "NTerm(St, NewEu, EuTail)" (par "procedure").
   % Unification avec la tête de la clause (e) de la grammaire.
   NewEu = Eu = [ae(id2, [ae(begin,t)])|[ie(q, [ae(begin,t)])]]

 % Appel du deuxième terme du corps  2-2
 ? process([ae(id2, [ae(begin,t)])|[ie(q, [ae(begin,t)])]], Su)
  % Unification avec la tête de la clause 3.
  Active = ae(id2, [ae(begin,t)])
  EuTail2 = [ie(q, [ae(begin,t)])]
  Su = [ae(id2, [ae(begin,t)])|SuTail]
  SuTail = ?

  % Appel du corps 3-1
  ? process([ie(q, [ae(begin,t)])], [ae(id2, [ae(begin,t)])|SuTail2])
   % Unification avec la tête de la clause 2.
   NTerm = q
   St = [ae(begin,t)]
   EuTail3 = []
   Su2 = [[ae(id2, [ae(begin,t)])]|SuTail2]

   % Appel du premier terme du corps  2-1
   ? procedure(q,[ae(begin,t)], NewEu2, [])
    % Appel du prédicat "NTerm(St, NewEu, EuTail)".
    % Unification avec la tête de la clause (f) de la grammaire.
    NewEu2 = Eu = [ae(id3, [ae(begin,t)])|[]]

   % Appel du deuxième terme du corps  2-2
   ? process([ae(id3, [ae(begin,t)])], [ae(id2, [ae(begin,t)])|SuTail2])
    % Unification avec la tête de la clause 3.
```

```
      Active = ae(id3, [ae(begin,t)])
      EuTail3 = []
      Su2 = [ae(id3, [ae(begin,t)])|SuTail2]
      SuTail2 = ?

      % Appel du corps 3-1
      ? process([], SuTail2)
        % Unification avec la tête de la clause 1.
        SuTail2 = []

   % Exit 3-1
     Su2 = [ae(id3, [ae(begin,t)])|SuTail2]
     SuTail2 = []

  % Exit  2-2
    Su = [ae(id2, [ae(begin,t)])|SuTail]
    SuTail = [ae(id3, [ae(begin,t)])]

% Exit but-1
  S1 = Su = [ae(id2, [ae(begin,t)], ae(id3, [ae(begin,t)])]

%%%%%%%
% Appel but-2
? process([ie(p,[ae(id2, [ae(begin,t)]), ae(id3, [ae(begin,t)])])], S2)
  % Unification avec la tête de la clause 2.
  NTerm = p
  St = [ae(id2, [ae(begin,t)]), ae(id3, [ae(begin,t)])]
  EuTail = []
  S2 = Su = ?

  % Appel du premier terme du corps  2-1
  ? procedure(p,[ae(id2, [ae(begin,t)]), ae(id3, [ae(begin,t)])], NewEu, [])
    % Appel du prédicat "NTerm(St, NewEu, EuTail)" (par "procedure").
    % Unification avec la tête de la clause (h) de la grammaire.
    Ss = [ae(begin,t)]
    StTail = [ae(id3, [ae(begin,t)])]
    EuTail = []
    NewEu = Eu = [ie(sp, [ae(begin,t)])|Eu1]
    Eu1 = ?

    % Appel du deuxième terme du corps  (h)-2
    ? p([ae(id3, [ae(begin,t)])], Eu1, [])
      % Unification avec la tête de la clause (i) de la grammaire.
      Ss = [ae(begin,t)]
      StTail = []
      EuTail = []
      Eu1 = [ie(sp, [ae(begin,t)])|Eu2]
      Eu2 = ?

      % Appel du deuxième terme du corps  (i)-2
      ? p([], Eu2, [])
        % Unification avec la tête de la clause (g) de la grammaire.
        Eu2 = []
```

```
  % Exit (h)-2
    Eu1 = [ie(sp, [ae(begin,t)])|Eu2]
    Eu2 = []

% Exit 2-1
  NewEu = Eu = [ie(sp, [ae(begin,t)])|Eu1]
  Eu1 =  [ie(sp, [ae(begin,t)])]


% Appel du deuxième terme du corps  2-2
? process([ie(sp, [ae(begin,t)]), ie(sp, [ae(begin,t)])], Su)
  % Unification avec la tête de la clause 2.
  NTerm = sp
  St = [ae(begin,t)]
  EuTail = [ie(sp, [ae(begin,t)])]
  Su = ?

  % Appel du premier terme du corps  2-1
  ? procedure(sp,[ae(begin,t)], NewEu, [ie(sp, [ae(begin,t)])])
    % Appel du prédicat "NTerm(St, NewEu, EuTail)".
    % Unification avec la tête de la clause (a) de la grammaire.
    NewEu = Eu = [ae(id1, [ae(begin,t)])|[ie(sp, [ae(begin,t)])]]

  % Appel du deuxième terme du corps  2-2
  ? process([ae(id1, [ae(begin,t)]), ie(sp, [ae(begin,t)])], Su)
    % Unification avec la tête de la clause 3.
    Active = ae(id1, [ae(begin,t)])
    EuTail2 = [ie(sp, [ae(begin,t)])]
    Su = [ae(id1, [ae(begin,t)])|SuTail]
    SuTail = ?

    % Appel du corps 3-1
    ? process([ie(sp, [ae(begin,t)])], Su2)
      % Unification avec la tête de la clause 2.
      NTerm = sp
      St = [ae(begin,t)]
      EuTail3 = []
      SuTail = Su2 = ?

      % Appel du premier terme du corps  2-1
      ? procedure(sp,[ae(begin,t)], NewEu2, [])
        % Appel du prédicat "NTerm(St, NewEu, EuTail)".
        % Unification avec la tête de la clause (a) de la grammaire.
        NewEu2 = Eu = [ae(id1, [ae(begin,t)])|[]]

      % Appel du deuxième terme du corps  2-2
      ? process([ae(id1, [ae(begin,t)])], Su2)
        % Unification avec la tête de la clause 3.
        Active = ae(id1, [ae(begin,t)])
        EuTail4 = []
        Su2 = [ae(id1, [ae(begin,t)])|SuTail2]
        SuTail2 = ?

        % Appel du corps 3-1
```

```
        ? process([], SuTail2)
          % Unification avec la tête de la clause 1.
          SuTail2 = []

       % Exit 2-2
         Su2 = [ae(id1, [ae(begin,t)])|SuTail2]
         SuTail2 = []

     % Exit  3-1
       SuTail = Su2 = [ae(id1, [ae(begin,t)])]

   % Exit 2-2
     Su = [ae(id1, [ae(begin,t)])|SuTail]
     SuTail =  [ae(id1, [ae(begin,t)])]

 % Exit  2-2
   Su = [ae(id1, [ae(begin,t)]), ae(id1, [ae(begin,t)])]

% Exit but-2
 S2 = Su = [ae(id1, [ae(begin,t)]), ae(id1, [ae(begin,t)])]

%%%%%%%
% Appel but-3
? process([ie(v, [ae(id1, [ae(begin,t)]), ae(id1, [ae(begin,t)])], S3)
 % Unification avec la tête de la clause 2.
 NTerm = v
 St = [ae(id1, [ae(begin,t)]), ae(id1, [ae(begin,t)])]
 EuTail = []
 S3 = Su = ?

 % Appel du premier terme du corps  2-1
 ? procedure(v, [ae(id1, [ae(begin,t)]), ae(id1, [ae(begin,t)])], NewEu, [])
  % Appel du prédicat "NTerm(St, NewEu, EuTail)" (par "procedure").
  % Unification avec la tête de la clause (k) de la grammaire.
  St = [ae(id1, [ae(begin,t)]), ae(id1, [ae(begin,t)])]
  EuTail = []
  NewEu = Eu = [ie(sv, [ae(id1, [ae(begin,t)]), ae(id1, [ae(begin,t)])])]

 % Appel du deuxième terme du corps  2-2
 ? process([ie(sv, [ae(id1, [ae(begin,t)]), ae(id1, [ae(begin,t)])])], Su)
  % Unification avec la tête de la clause 2.
  NTerm = sv
  St = [ae(id1, [ae(begin,t)]), ae(id1, [ae(begin,t)])]
  EuTail = []
  Su = ?

  % Appel du premier terme du corps  2-1
  ? procedure(sv, [ae(id1, [ae(begin,t)]), ae(id1, [ae(begin,t)])], NewEu, [])
   % Appel du prédicat "NTerm(St, NewEu, EuTail)".
   % Unification avec la tête de la clause (c) de la grammaire.
   Ss = [ae(begin,t)]
   StTail = ae(id1, [ae(begin,t)])
   NewEu = Eu = [ie(s, [ae(begin,t)])|Eu1]
   Eu1 = ?
```

```
  % Appel du deuxième terme du corps  (c)-2
  ? sv([ae(id1, [ae(begin,t)])], Eu1, [])
    % Unification avec la tête de la clause (c) de la grammaire.
    Ss = [ae(begin,t)]
    StTail = []
    EuTail = []
    Eu1 = Eu = [ie(s, [ae(begin,t)])|Eu2]
    Eu2 = ?

    % Appel du deuxième terme du corps  (c)-2
    ? p([], Eu2, [])
      % Unification avec la tête de la clause (b) de la grammaire.
      Eu2 = []

  % Exit (c)-2
    Eu1 = [ie(s, [ae(begin,t)])|Eu2]
    Eu2 = []

% Exit 2-1
  NewEu = Eu = [ie(s, [ae(begin,t)])|Eu1]
  Eu1 = [ie(s, [ae(begin,t)])]


% Appel du deuxième terme du corps  2-2
? process([ie(s, [ae(begin,t)]), ie(s, [ae(begin,t)])], Su)
  % Unification avec la tête de la clause 2.
  NTerm = s
  St = [ae(begin,t)]
  EuTail = [ie(s, [ae(begin,t)])]
  Su = ?

  % Appel du premier terme du corps  2-1
  ? procedure(s,[ae(begin,t)], NewEu, [ie(s, [ae(begin,t)])])
    % Appel du prédicat "NTerm(St, NewEu, EuTail)".
    % Unification avec la tête de la clause (m) de la grammaire.
    EuTail = [ie(s, [ae(begin,t)])]
    StTail = []
    NewEu = Eu = [ae(idend, t)|Eu1]
    Eu1 = ?

    % Appel du deuxième terme du corps  (m)-2
    ? s([], Eu1, [ie(s, [ae(begin,t)])])
      % Unification avec la tête de la clause (l) de la grammaire.
      Eu1 = Eu =  [ie(s, [ae(begin,t)])]

  % Exit 2-1
    NewEu = Eu = [ae(idend, t)|Eu1]
    Eu1 = [ie(s, [ae(begin,t)])]

  % Appel du deuxième terme du corps  2-2
  ? process([ae(idend, t), ie(s, [ae(begin,t)])], Su)
    % Unification avec la tête de la clause 3.
    Active = ae(idend, t)
```

```
  EuTail2 = [ie(s, [ae(begin,t)])]
  Su = [ie(s, [ae(begin,t)])|SuTail]
  SuTail = ?

  % Appel du corps 3-1
  ? process([ie(s, [ae(begin,t)])], Su2)
    % Unification avec la tête de la clause 2.
    NTerm = s
    St = [ae(begin,t)]
    EuTail = []
    SuTail = Su2 = ?

    % Appel du premier terme du corps  2-1
    ? procedure(s,[ae(begin,t)], NewEu, [])
      % Appel du prédicat "NTerm(St, NewEu, EuTail)".
      % Unification avec la tête de la clause (m) de la grammaire.
      EuTail = []
      StTail = []
      NewEu = Eu = [ae(idend, t)|Eu1]
      Eu1 = ?

      % Appel du deuxième terme du corps  (m)-2
      ? s([], Eu1, [])
          % Unification avec la tête de la clause (l) de la grammaire.
             Eu1 = Eu =  []

    % Exit 2-1
      NewEu = Eu = [ae(idend, t)|Eu1]
      Eu1 = []

    % Appel du deuxième terme du corps  2-2
    ? process([ae(idend, t)], Su2)
      % Unification avec la tête de la clause 3.
      Active = ae(idend, t)
      EuTail3 = []
      Su2 = [ae(idend, t)]|SuTail]
      SuTail = ?

      % Appel du deuxième terme du corps  2-1
      ? process([], SuTail)
        % Unification avec la tête de la clause 1.
        SuTail = []

    % Exit 2-2
      Su2 = [ae(idend, t)]|SuTail]
      SuTail = []

  % Exit  3-1
    SuTail = Su2 = [ae(idend, t)]

% Exit 2-2
  Su = [ie(s, [ae(begin,t)])|SuTail]
  SuTail = [ae(idend, t)]
```

```
   % Exit  2-2
     Su = [ie(s, [ae(begin,t)]), ae(idend, t)])]

  % Exit  2-2
    Su = [ie(s, [ae(begin,t)]), ae(idend, t)])]

% Exit but-3
  S3 = Su = [ie(s, [ae(begin,t)]), ae(idend, t)])]

%%%%%%%
% Appel but-4
? fin([ie(s, [ae(begin,t)]), ae(idend, t)])])
  % Unification avec la tête de la clause 5.
  SmTail = [ae(idend, t)]

  % Appel du premier terme du corps  5-1
  ? procedure_fin
    % Affichage du résultat

  % Appel du deuxième terme du corps  5-2
  ? fin([ae(idend, t)])
    % Unification avec la tête de la clause 5.
    SmTail = []

    % Appel du premier terme du corps  5-1
    ? procedure_fin
      % Affichage du résultat

    % Appel du deuxième terme du corps  5-2
    ? fin([])
      % Unification avec la tête de la clause 4.

    % Exit 5

  % Exit 5

% Exit but-4
```

5.2.9 Possibilité d'amélioration d'algorithme

Nous présentons dans cette dernière section une possibilité d'amélioration algorithmique en s'appuyant sur l'article publié par les créateurs du système SAX [23].

Cette amélioration de l'algorithme de *Bottom-up Chart Parsing* propose la réduction de la création des arcs inutiles. En effet, une des principales sources de l'inefficacité de l'analyse syntaxique se trouve dans les ambiguïtés syntaxiques, c'est-à-dire les possibilités multiples d'interprétation de structures telles que celles entre qualifiant et qualifié. Ce problème se retrouve dans les analyses réalisées par un algorithme utilisant un chart au travers de la présence d'un grand nombre d'arcs ne constituant pas une représen-

tation correcte de la structure de la phrase. Illustrons cette ambiguïté avec l'exemple de résultat d'analyse présenté dans [23], avec la grammaire :

$SV \rightarrow SN\ SV$ $\quad$ $V \rightarrow$ 壊れた | 修理した[4]
$SN \rightarrow Sub\ P$ $\quad$ $Sub \rightarrow$ 車 | 太郎[5]
$N \rightarrow SV\ Sub$ $\quad$ $P \rightarrow$ は | を[6]
$SV \rightarrow V$

et la phrase d'entrée :

太郎$_1$は$_2$壊れた$_3$車$_4$を$_5$修理した$_6$
(Tarô a réparé la voiture cassée.)

L'analyse de cette phrase avec la grammaire ci-dessus entraîne la création des arcs listés dans le tableau 5.1 (voir page suivante)[7][8].

Parmi ces arcs, ceux nécessaires pour générer le résultat final sont marqués par le symbole ◯ dans la colonne la plus à droite. Ce tableau montre que même pour une phrase simple, l'analyse normale génère un grand nombre d'arcs inutiles. Les arcs non nécessaires représentent 38% du total (11/29), et si on ne tient pas compte des arcs non actifs représentant les *tango*, leur proportion s'élève à 52% (11/21).

Comme méthodes plus efficaces, nous connaissons les techniques d'analyse syntaxique utilisant des grammaires non-contextuelles probabilistes, qui permettent d'obtenir en priorité le résultat le plus probable, en s'appuyant la probabilité attribuée à chaque règle de grammaire.

La méthode proposée par les créateurs de SAX utilise les informations statistiques portées par les rapports dépendantiels entre les mots, *kakari-uke*, pour contrôler l'application des règles de grammaire. Elle sépare les informations statistiques de la grammaire afin d'obtenir une plus haute portabilité de la grammaire. En effet, avec cette solution, le changement de domaine des textes à analyser ne nécessite que l'apprentisage des probabilités de chaque rapport *kakari-uke*, sans modification de la grammaire elle-même, tandis que les méthodes avec une grammaire non-contextuelle probabiliste nécessitent l'apprentisage des probabilités de la grammaire correspondante.

Nous allons maintenant étudier comment cette amélioration est réalisée concrètement.

Mécanisme d'amélioration

En utilisant les probabilités attribuées à chaque rapport entre les *tango*, on obtient pour la phrase d'exemple les informations de *kakari-uke* suivantes :

[4] 壊れた (*kowareta*, casser (passé)), 修理した (*shûrishita*, réparer (passé))
[5] 車 (*kuruma*, voiture), 太郎 (*tarô*, [nom propre])
[6] は (*wa*, [marqueur de thème]), を (*wo*, [marqueur de COD])
[7] « Dép. » représente l'élément dont dépendra le syntagme traité. Nous parlons de la signification des chiffres inscrits dans cette case plus loin.
[8] Les substantifs sont étiquetés par *n*.

No.	Posit			Arc	
	Début	Fin	Dép.		
(1)	0	1		$[太郎]_n$	○
(2)	0	1	2	$[[太郎]_n[?]_p]_{sn}$	○
(3)	1	2		$[は]_p$	○
(4)	0	2		$[[太郎]_n[は]_p]_{sn}$	○
(5)	0	2	6, 3	$[[[太郎]_n[は]_p]_{sn}[?]_{sv}]_{sv}$	○
(6)	2	3		$[壊れた]_v$	○
(7)	2	3		$[壊れた]_{sv}$	○
(8)	2	3	4	$[[壊れた]_{sv}[?]_n]_n$	○
(9)	0	3		$[[[太郎]_n[は]_p]_{sn}[壊れた]_{sv}]_{sv}$	×
(10)	0	3		$[[[[太郎]_n[は]_p]_{sn}[壊れた]_{sv}]_{sv}[?]_n]_n$	×
(11)	3	4		$[車]_n$	○
(12)	3	4		$[[車]_n[?]_p]_{sn}$	△
(13)	2	4		$[[壊れた]_{sv}[車]_n]_n$	○
(14)	2	4	5	$[[壊れた]_{sv}[車]_n]_n[?]_p]_{sn}$	○
(15)	0	4		$[[[[太郎]_n[は]_p]_{sn}[壊れた]_{sv}]_{sv}[車]_n]_n$	×
(16)	0	4		$[[[[[太郎]_n[は]_p]_{sn}[壊れた]_{sv}]_{sv}[車]_n]_n[?]_p]_{sn}$	×
(17)	4	5		$[を]_p$	○
(18)	3	5		$[[車]_n[を]_p]_{sn}$	△
(19)	3	5		$[[[車]_n[を]_p]_{sn}[?]_{sv}]_{sv}$	△
(20)	2	5		$[[[壊れた]_{sv}[車]_n]_n[を]_p]_{sn}$	○
(21)	2	5	6	$[[[[壊れた]_{sv}[車]_n]_n[を]_p]_{sn}[?]_{sv}]_{sv}$	○
(22)	0	5		$[[[[[太郎]_n[は]_p]_{sn}[壊れた]_{sv}]_{sv}[車]_n]_n[を]_p]_{sn}$	×
(23)	0	5		$[[[[[[太郎]_n[は]_p]_{sn}[壊れた]_{sv}]_{sv}[車]_n]_n[を]_p]_{sn}[?]_{sv}]_{sv}$	×
(24)	5	6		$[修理した]_v$	○
(25)	5	6		$[修理した]_{sv}$	○
(26)	3	6		$[[[車]_n[を]_p]_{sn}[修理した]_{sv}]_{sv}$	△
(27)	2	6		$[[[[壊れた]_{sv}[車]_n]_n[を]_p]_{sn}[修理した]_{sv}]_{sv}$	○
(28)	0	6		$[[[太郎]_n[は]_p]_{sn}[[[[壊れた]_{sv}[車]_n]_n[を]_p]_{sn}[修理した]_{sv}]_{sv}]_{sv}$	○
(23)	0	6		$[[[[[[太郎]_n[は]_p]_{sn}[壊れた]_{sv}]_{sv}[車]_n]_n[を]_p]_{sn}[修理した]_{sv}]_{sv}$	×

Tab. 5.1 – Arcs créés à chaque étape d'analyse

$$
\begin{aligned}
1 &\rightarrow 2 \\
2 &\xrightarrow{,1} 3 \\
2 &\xrightarrow{,9} 6 \\
3 &\rightarrow 4 \\
4 &\rightarrow 5 \\
5 &\rightarrow 6
\end{aligned}
$$

Les chiffres aux deux extrémités des flèches sont les numéros correspondant à chaque *tango* de la phrase d'entrée, attribués suivant leur ordre d'apparition. Par exemple, 1 correspond à 太郎, 2 à は, et ainsi de suite. Les chiffres au dessus des flèches représentent la probabilité de dépendance du *tango* gauche vis à vis du *tango* droit. L'absence de chiffre représente une probabilité de 1.

En utilisant ces informations, on peut contrôler l'application des règles. Lorsque l'initialisation ou la complétion d'arc est possible, on examine si sa réalisation n'est pas inutile en s'appuyant sur les informations de *kakari-uke*.

La génération des arcs marqués par × dans le tableau 5.1 page précédente peut être évitée par l'annulation de certaines complétions inutiles, et celle des arcs marqués par △, par l'annulation de certaines initialisations. Dans l'exemple, le rapport dépendentiel entre les *tango* est utilisé pour simplifier l'explication. En pratique, on utilise le rapport entre les syntagmes *bunsetsu*[9]. En général, la probabilité du rapport entre les *tango* constituant les syntagmes *bunsetsu*, est considérée comme égale à 1.

Contrôle des complétions d'arc

La complétion d'un arc actif par un arc non actif est équivalente à la constitution d'une relation de dépendance. Il est donc possible d'éviter l'application des complétions inutiles en utilisant les informations de *kakari-uke*. Il s'agit en fait de donner aux arcs actifs des informations sur le(s) élément(s) dont ils dépendent[10]. Lors de la complétion, on vérifie alors tout simplement si l'arc complétant le terme non défini le plus à gauche correspond à cet élément.

En réformulant ce constat, on ajoute alors la condition supplémentaire suivante pour l'application de la complétion :

> **Condition supplémentaire pour la complétion :** la position de l'élément de dépendance de l'arc actif et la position finale de l'arc non actif doivent être identiques.

Dans le tableau 5.1 page précédente, la complétion de l'arc (2) par l'arc (3) est réalisée car la position de l'élément de dépendance de l'arc (2) et la position finale de l'arc non actif (3) sont toutes les deux 2. En revanche, l'arc (5) et l'arc (8) ne satisfaisant pas cette dernière condition, leur complétion ne se réalise pas. Ainsi, il est possible de maîtriser la génération d'arcs inutiles.

Contrôle des initialisations d'arc

Réfléchissons aux situations où l'initialisation génère un arc inutile.

Supposons qu'il y ait une séquence de trois mots, A, B et C, et que A dépende de B et B dépende de C. Dans ce cas, on doit créer un sous-arbre D à partir de A et B, et constituer un sous-arbre E à partir de D et C. Mais la constitution d'un sous-arbre F à partir de B et C est inutile car il ne sera pas utilisé plus tard.

La situation dans laquelle la constitution du sous-arbre F n'est pas inutile, est le cas où rien ne dépend de B, tel que le cas où A et B dépendent tous les deux de C.

[9] Le système d'analyse des rapports de *kakari-uke* développé par [17] est utilisé pour cette opération.

[10] Dans le tableau 5.1 page précédente, l'élément dont l'arc actif courant dépend, est indiqué dans la case « Dép. ».

On peut déduire de ce que nous avons vu, que l'initialisation d'un arc non actif est utile lorsque rien ne dépend de lui, autrement dit, quand il ne peut compléter aucun arc actif précédent. Par exemple, dans le tableau 5.1 page 109, l'arc (12) ne serait pas généré si un contrôle sur les initialisations était effectué, car l'arc (8) dépendant de l'arc (11), on ne réalise pas l'initialisation de ce dernier.

En réformulant ce constat, on ajoute alors la condition supplémentaire suivante pour l'application de l'initialisation :

> **Condition supplémentaire pour l'initialisation :** l'initialisation est applicable si et seulement si rien ne dépend de l'arc non actif à initialiser.

Ainsi, il est possible de maîtriser les initialisations générant des arcs inutiles.

L'article [23] présente le résultat de l'expérience de cette méthode appliquée sur le système SAX. D'après ce rapport, l'utilisation des informations de *kakari-uke* réduit, par rapport au système utilisant uniquement la grammaire simple fournie avec le système (env. 150 règles), le nombre d'arcs créés de 45% et rend la durée de traitement 8,4 fois inférieure pour une phrase relativement simple. Pour une phrase assez longue, elle réduit le nombre d'arcs créés de 22% et rend la durée de traitement 11,9 fois inférieure.

5.3 MSLR

5.3.1 Introduction

L'analyseur morpho-syntaxique MSLR (*Morphological and Syntactic LR Parser*) a été développé dans un laboratoire de l'Institut Technologique de Tokyo.

L'analyseur MSLR génère des tables d'analyse LR et un index de dictionnaire. Les tables d'analyse LR sont créées à partir d'une grammaire hors contexte et d'une table de connexion (la grammaire et la table de connexion peuvent être fournies par l'utilisateur, mais le système met ses propres grammaire et table à disposition des utilisateurs). L'index de dictionnaire est créé à partir d'un dictionnaire. Là encore, l'utilisateur peut choisir entre son propre dictionnaire et celui fourni avec le système.

En utilisant ces deux outils – les tables d'analyse LR et l'index de dictionnaire – créés par le système, le MSLR Parser réalise une analyse morpho-syntaxique du texte entré. L'utilisateur de cet analyseur a également la liberté de choisir la méthode d'analyse morphologique. En effet, le système peut ne réaliser qu'une analyse syntaxique en laissant à l'utilisateur la possibilité d'utiliser l'analyseur morphologique de son choix. Dans ce cas, l'utilisateur fournit le résultat de l'analyse morphologique comme donnée d'entrée.

Avec une telle liberté dans le choix de la grammaire ainsi que dans celui de l'analyse morphologique, MSLR Parser est capable d'analyser n'importe

quelle langue, ne se limitant donc pas au japonais.

L'analyseur MSLR est caractérisé d'autre part par l'utilisation de procédures parallèles d'analyses morphologique et syntaxique. Ce qui permet, d'après ses créateurs, une meilleure efficacité par rapport aux systèmes réalisant chaque analyse indépendamment.

Par ailleurs, cet analyseur est, comme son nom l'indique, caractérisé par l'utilisation de l'algorithme d'analyse syntaxique LR que nous avons étudié dans la section 4.3 page 68. Comme nous l'avons vu dans cette étude, l'algorithme LR est caractérisé par le fait que le programme moteur étant le même pour tous les analyseurs LR, seules les tables d'analyse changent d'un analyseur à l'autre. Nous allons donc nous intéresser non pas aux détails de l'implantation de cet algorithme mais plutôt aux caractéristiques de la construction des tables d'analyse et à la grammaire, qui est à la base de ces tables d'analyse. Cette étude permettra de mieux comprendre les particularités ou les avantages de cet analyseur.

Cette section, consacrée à l'analyse du MSLR Parser, est constituée de l'étude de l'architecture du système MSLR, de ses caractéristiques, et enfin de la grammaire japonaise intégrée dans le système.

5.3.2 Architecture

Le MSLR Parser est accompagné de deux outils : un outil de création des tables d'analyse LR et un outil de création de l'index de dictionnaire.

Le premier outil génère des tables d'analyse LR à partir d'une grammaire CFG et d'une table de connexion. En général, le système charge les tables d'analyse en mémoire centrale pour analyser une phrase. Cependant, lors de l'utilisation d'une grammaire de taille importante, il est possible de créer un index des tables d'analyse LR pour éviter de les stocker en totalité dans la mémoire, qui peut être insuffisante pour cette grammaire. Dans ce cas, le MSLR Parser réalise l'analyse en consultant les tables LR présentes sur le disque, comme pour le dictionnaire.

Le second outil crée un index de dictionnaire à partir d'un dictionnaire.

En utilisant ces deux ensembles de données – les tables d'analyse LR et l'index de dictionnaire – créés par le système, le MSLR Parser réalise l'analyse morpho-syntaxique du texte entré. Ce mécanisme d'analyse est représenté sur la figure 5.2 (voir page suivante).

Nous allons voir à présent différentes caractéristiques de ce système.

5.3.3 Composant de création des tables d'analyse, Table de connexion

L'outil utilisé pour la création des tables d'analyse LR est capable de créer trois types de tables d'analyse LR, à savoir des tables SLR, des tables LR Canonique et des tables LALR. Mais sa particularité principale se trouve

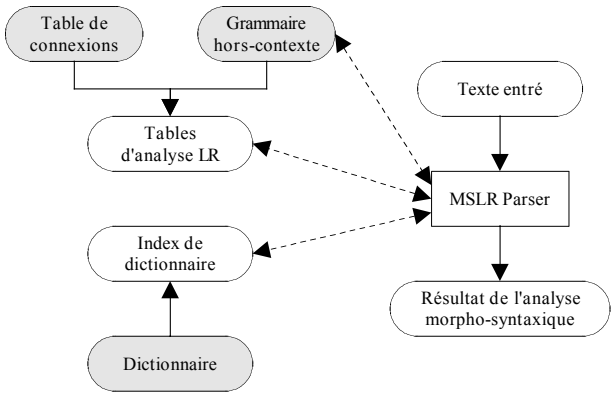


FIG. 5.2 – Architecture d'un analyseur MSLR

dans le fait qu'il peut créer des tables d'analyse tenant compte des conditions de connexion entre deux catégories morphologiques définies dans la table de connexion.

En prenant en compte ces conditions de connexion entre deux morphèmes dans les tables d'analyse LR, le système a réussi à intégrer les contraintes du niveau syntaxique et du niveau morphologique à l'intérieur des tables d'analyse LR, permettant ainsi de réaliser l'unification des analyses morphologique et syntaxique sans modifier l'analyseur lui-même.

Le système MSLR définit des sous-catégories morphologiques – dites 細品詞 (*saihinshi*) –, obtenues par décomposition des catégories morphologiques selon leurs caractères de connexion à droite et à gauche. La description de ces caractères de connexion est basée sur la description du dictionnaire EDR [10], fourni avec l'analyseur comme dictionnaire du système. Si l'on définissait une grammaire en tenant compte du niveau de ces sous-catégories morphologiques, le nombre de productions, et donc la charge de travail humaine, serait très élevé. Le système MSLR définit alors comme conditions de connexion des *tango* les conditions de connexion entre deux sous-catégories morphologiques, pour qu'au niveau syntaxique il soit possible de ne pas traiter chacune de ces sous-catégories morphologiques. Dans la grammaire, ces sous-catégories morphologiques sont transformées en catégories morphologiques par les productions appelées « Règles des sous-catégories morphologiques » (細品詞規則, *saihinshi-kisoku*). La grammaire peut donc ne traiter que les niveaux à partir des catégories morphologiques, sans se préoccuper des sous-catégories.

Les conditions de connexion définies dans la table de connexion permettent de ne pas inscrire dans les tables d'analyse les actions générant un arbre qui enfreint les conditions de connexion. Ainsi, avec des tables tenant compte des contraintes définies dans la table de connexion, on ne peut obtenir comme résultat d'analyse un arbre enfreignant les conditions de connexion.

Nous allons maintenant nous intéresser à un exemple concret de l'effet

produit par la prise en compte des contrainte de connexion dans les tables d'analyse, présenté dans le manuel de MSLR Parser [51].

Exemple 15 Considérons la phrase :

かおるにあいます (*ka o ru ni a i ma su*, je rencontre Kaoru.)

Les résultats obtenus avec les tables d'analyse sans tenir compte des conditions de connexion sont les suivants [11] :

[S, [SV, [SP, [SN, [sub, [nom propre, *ka o ru*]]],
[particule, [particule de cas, *ni*]]], [SV, [vb, [rad vb, [rad vb5 sur wa, *a*]],
[term, [term vb5 mdf en i, *i*]]], [aux, [aux, *masu*]]]]
[S, [SV, [SP, [SN, [sub, [nom propre, *ka o ru*]]],
[particule, [particule de cas, *ni*]]], [SV, [vb, [rad vb, [rad vb5 sur ka, *a*]],
[term, [term vb5 mdf en i, *i*]]], [aux, [aux, *masu*]]]]
[S, [SV, [SP, [SN, [sub, [nom propre, *ka o ru*]]],
[particule, [particule de cas, *ni*]]], [SV, [vb, [rad vb, [rad vb5 sur wa, *a*]],
[term, [term vb5 en i, *i*]]], [aux, [aux, *masu*]]]]
[S, [SV, [SP, [SN, [sub, [nom propre, *ka o ru*]]],
[particule, [particule de cas, *ni*]]], [SV, [vb, [rad vb, [rad vb5 sur ka, *a*]],
[term, [term vb5 en i, *i*]]], [aux, [aux, *masu*]]]]
[S, [SV, [SAdv, [SV,
[vb, [rad vb, [rad vb5 sur ra, *ka o*]], [term, [term vb5 en ru, *ru*]]]],
[particule, [particule de cas, *ni*]]], [SV, [vb, [rad vb, [rad vb5 sur wa, *a*]],
[term, [term vb5 mdf en i, *i*]]], [aux, [aux, *masu*]]]]
[S, [SV, [SAdv, [SV,
[vb, [rad vb, [rad vb5 sur ra, *ka o*]], [term, [term vb5 en ru, *ru*]]]],
[particule, [particule de cas, *ni*]]], [SV, [vb, [rad vb, [rad vb5 sur ka, *a*]],
[term, [term vb5 mdf en i, *i*]]], [aux, [aux, *masu*]]]]
[S, [SV, [SAdv, [SV,
[vb, [rad vb, [rad vb5 sur ra, *ka o*]], [term, [term vb5 en ru, *ru*]]]],
[particule, [particule de cas, *ni*]]], [SV, [vb, [rad vb, [rad vb5 sur wa, *a*]],
[term, [term vb5 en i, *i*]]], [aux, [aux, *masu*]]]]
[S, [SV, [SAdv, [SV,
[vb, [rad vb, [rad vb5 sur ra, *ka o*]], [term, [term vb5 en ru, *ru*]]]],
[particule, [particule de cas, *ni*]]], [SV, [vb, [rad vb, [rad vb5 sur ka, *a*]],
[term, [term vb5 en i, *i*]]], [aux, [aux, *masu*]]]]

Avec les tables d'analyse tenant compte des conditions de connexion, on obtient le seul résultat suivant :

[S, [SV, [SP, [SN, [sub, [nom propre, *ka o ru*]]],

[11] Conventions de notation : rad = radical, sub = substantif, vb = verbe, term = terminaison, rad vb5 sur wa = ワ行五段動詞語幹 (radical d'un verbe de type *5 dan* sur la colonne *wa*), term vb5 mdf en i = 語尾五段音便い (terminaison d'un verbe de type *5 dan* avec modification en *i*)

[particule, [particule de cas, *ni*]]], [SV, [vb, [rad vb, [rad vb5 sur wa, *a*]], [term, [term vb5 en i, *i*]]], [aux, [aux, *masu*]]]]]

Cette restriction provient des deux conditions de connexion suivantes, prises en compte dans les tables d'analyse :

1. La connexion entre la terminaison d'un verbe de type *5 dan* en *ru* et une particule de cas est impossible.
2. Lorsque « *a* » est suivi par « *i* », il n'y a qu'une seule connexion possible, à savoir la connexion entre le radical d'un verbe de type *5 dan* sur *wa* et la terminaison d'un verbe de type *5 dan* en *i*.

5.3.4 Hiérarchisation des résultats

Le système propose deux façons différentes de hiérarchiser les résultats.

La première méthode consiste à utiliser le *Probabilistic Generalized LR Model* (modèle PGLR ci-après). En fait, le système dispose de plusieurs fonctions :

- apprentissage des modèles PGLR ;
- calcul de la probabilité de génération des arbres syntaxiques par les modèles PGLR ;
- production comme résultat, uniquement des n arbres syntaxiques les plus élévés en termes de probabilité telle que décrite précédemment.

Les modèles PGLR sont des modèles statistiques donnant la probabilité de génération d'un arbre syntaxique.

Lorsqu'on réalise une analyse avec les modèles PGLR, le système donne comme résultats les dix arbres syntaxiques les plus élevés pour leur probabilité de génération, et ce dans l'ordre décroissant. Le nombre d'arbres de résultat peut également être défini par l'utilisateur. Sur l'écran de résultat, la probabilité est affichée derrière son arbre syntaxique.

Le système peut afficher également comme résultat d'une analyse les actions des tables d'analyse utilisées et le nombre d'utilisations de ces actions. Lorsqu'à partir d'une même phrase, plusieurs arbres d'analyse sont générés, le nombre d'utilisations d'une action est calculé comme « 1/nombre d'arbres générés ». Par exemple, si la phrase génère 100 arbres d'analyse, la fréquence de chaque action est comptée comme $\frac{1}{100}$.

La seconde méthode repose sur les tables d'analyse LR fournies avec le système. En effet, elles permettent de réaliser la hiérarchisation des résultats par deux méthodes heuristiques : la méthode du nombre minimum de morphèmes et la méthode du nombre minimum de syntagmes (*bunsetsu*). Ainsi, l'analyseur MSLR peut fournir comme résultat d'analyse les arbres syntaxiques dans l'ordre suivant les règles ci-dessous :

1. sortir les arbres d'analyse selon leur nombre de morphèmes dans l'ordre croissant ;

2. sortir les arbres ayant le même nombre de morphèmes selon leur nombre de syntagmes *bunsetsu* dans l'ordre croissant.

La méthode du nombre minimun de morphèmes est réalisée par affectation de probabilités d'exécution aux actions « réduire par une production dont la partie droite est une catégorie morphologique ». La méthode du nombre minimun de syntagmes est réalisée par affectation de probabilités d'exécution aux actions `réduire` créées par une production constituant un syntagme *bunsetsu*. Le score donné par le système à chaque arbre correspond au produit de ces pseudo-probabilités affectées aux actions. Ces scores étant donnés uniquement pour hiérarchiser les résultats, les scores eux-même n'ont pas de sens contrairement aux probabilités des modèles PGLR appris par le corpus.
L'exemple de résultat d'analyse de la phrase 通常生活に戻った (*tsû jô sei katsu ni modo t ta*, je suis revenu à une vie normale) avec cette fonction de hiérarchisation est présenté dans le tableau ci-dessous (issu de [51]).

Nombre de morphèmes	Nombre de *bunsetsu*	Score	Résultat d'analyse
7	2	4.955931e-01	(通常/生活/に)(戻/っ/た/。)
7	3	4.950978e-01	(通常)(生活/に)(戻/っ/た/。)
8	2	4.484312e-01	(通/常/生活/に)(戻/っ/た/。)
8	3	4.479830e-01	(通常/生)(活/に)(戻/っ/た/。)
9	2	4.057573e-01	(通/常/生/活/に)(戻/っ/た/。)
9	3	4.053518e-01	(通/常)(生/活/に)(戻/っ/た。)

Dans les phrases du tableau ci-dessus, « / » représente la frontière entre les morphèmes et « (» et «) » la frontière entre les *bunsetsu*. Le résultat correct : (通常, normal /生活, vie /に, [destination])(戻, v. revenir (rad.) /っ, partie variante /た, [passé] /。, [point final]), ayant le moins de morphèmes et de *bunsetsu*, apparaît bien en tête de la liste.

5.3.5 Grammaire fournie avec le système

La grammaire fournie avec le système est une grammaire hors contexte pour l'analyse du japonais. Cette grammaire est capable de générer des arbres syntaxiques marquant les limites des syntagmes *bunsetsu* et leur relation de dépendance[12].

Sur la figure 5.3 (voir page suivante) est présenté un exemple – tiré du manuel [51] – de résultat d'analyse par cette grammaire de la phrase :

[12]Dans cette partie, la notion de dépendance est non pas celle introduite par Tesnière, mais la notion linguistique japonaise, « *kakari-uke* », que nous avons étudiée dans la section 1.4 page 11.

MSLRパーザ	は、	形態素解析	と
MSLR Parser	[particule marquant le thème]	analyse morphologique	et

構文解析	を	同時に	行います。
analyse syntaxique	[particule marquant le COD]	en même temps	réaliser

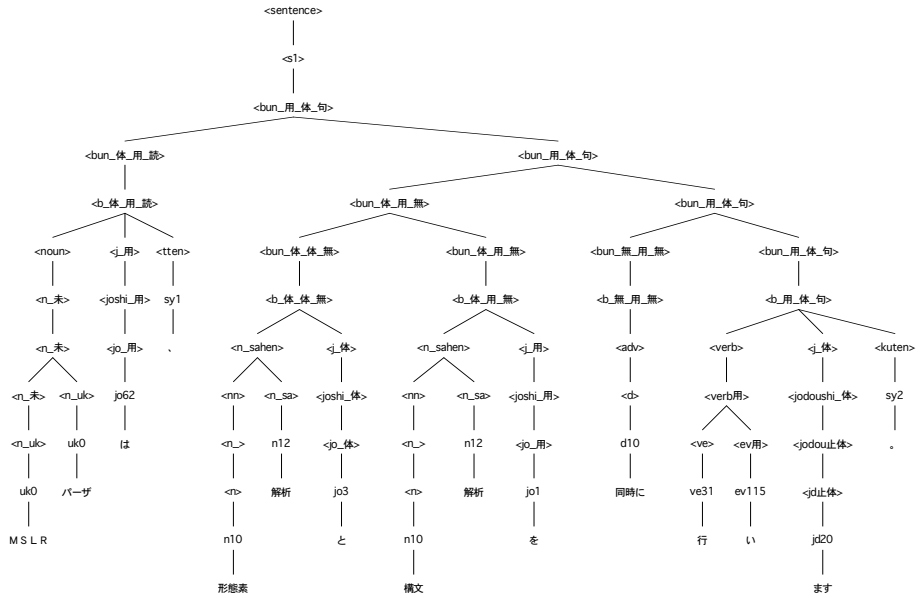


FIG. 5.3 – Arbre syntaxique résultat

Les non-terminaux représentant des constituants plus grands que les syntagmes *bunsetsu* et les *bunsetsu* sont dénotés sous forme de quadruplet comme <a_(X)_(Y)_(Z)>.
Chaque symbole porte l'information suivante :

(a) Le type de syntagme : b représente les syntagmes *bunsetsu*, bun, les syntagmes plus larges, et *s*, la phrase.

(X) La propriété de réception du syntagme : il existe quatre valeurs. 用 (*yô*) signifie que le syntagme peut recevoir une qualification portant sur les *yôgen* (verbes et qualificatifs), 体 (*tai*) signifie qu'il peut recevoir une qualification portant sur les *taigen* (substantifs), 用体 (*yôtai*) signifie qu'il peut recevoir les deux, 無 (*mu*) signifie qu'il ne reçoit aucune qualification.

(Y) La propriété de dépendance du syntagme : il existe quatre valeurs. 用 (*yô*) signifie que le syntagme peut être une qualification portant sur les *yôgen*, 体 (*tai*) signifie qu'il peut être une qualification portant sur les

taigen, 用体 (*yôtai*) signifie qu'il peut jouer les deux, 無 (*mu*) signifie qu'il ne joue aucune qualification.

(Z) Présence ou non d'un symbole de ponctuation : il existe trois valeurs. 読 (*tô*) signifie que le syntagme contient une virgule, 句 (*ku*) signifie qu'il contient un point final et 無 (*mu*) signifie qu'il ne contient aucune ponctuation.

La relation de dépendance entre les différents syntagmes est représentée dans l'arbre binaire dans lequel les non-terminaux représentant les syntagmes sont ses feuilles. Le syntagme représenté par la feuille la plus à droite gouvernée par la branche gauche de la racine dépend du syntagme représenté par la feuille la plus à droite gouvernée par la branche droite de la racine. Ainsi, dans l'arbre syntaxique d'exemple, on peut constater les relations de dépendance suivantes :

(Syntagme dépendant)		(Syntagme récepteur)
MSLRパーザは、	→	行います。
形態素解析と	→	構文解析を
構文解析を	→	行います。
同時に	→	行います。

Par ailleurs, la grammaire peut générer un arbre dans lequel un substantif constitue un syntagme *bunsetsu*. Par exemple, avec un ensemble de règles telles que :

<b_体_体_無> - -> <noun>
<noun> - - > <nn>
<nn> - - > <n_>
<n_> - - > <n>

on obtient la dérivation :

<b_体_体_無> ⇒ <noun> ⇒ <nn> ⇒ <n_> ⇒ <n>

Ce qui augmente les ambiguïtés syntaxiques concernant la combinaison des unités lors de l'analyse d'une phrase dans laquelle plusieurs substantifs sont juxtaposés. Le système a résolu ce problème en modifiant les tables d'analyse pour que plusieurs substantifs contigus constituent un seul syntagme. En fait, les actions `réduire` créées par les productions dans lesquelles un substantif constitue un syntagme, ont été supprimées des tables LR lorsque le symbole de pré-vision est un substantif.

Par ailleurs, lors d'une analyse avec la grammaire du système, il est possible d'appeler un sous-programme traitant les mots inconnus. Le traitement des mots inconnus, c'est-à-dire des mots corrects mais absents du dictionnaire – notamment les néologismes ou les noms propres –, est une des préoccupations majeures de l'analyse des langues.

Dans la grammaire du système, les mots inconnus appartiennent à une catégorie appelée « uk0 ». Le MSLR Parser traite ainsi les séquences de caractères considérées comme des mots inconnus, comme des mots de la catégorie « uk0 », permettant ainsi d'éviter toute interruption due aux séquences non reconnues. Dans la grammaire du système, les mots inconnus ont un statut à peu près équivalent à celui des noms propres. L'existence des mots inconnus est donc supposée nulle pour le système.

Reconnaissance des mots inconnus uk0 Le programme reconnaît les mots inconnus suivant les règles ci-dessous :

1. Considérer la séquence la plus longue de caractères *katakana*, de lettres (alphabétiques), de chiffres ou de symboles comme un mot inconnu.
2. Lorsqu'on ne trouve comme mot candidat que l'unité constituée d'un seul *kanji*, considérer la séquence la plus longue de kanji comme un mot inconnu.

Exemple 16 Voici des exemples pour lesquels les règles ci-dessus sont respectivement appliquées.

1. 名前はタナカさんと言います。 (*na mae ha ta na ka sa n to i i ma su*, « son nom est Tanaka »)
   ⇒ On considère la séquence de trois *katakana* « タナカ », comme un mot inconnu.
2. 某国の対鑑ミサイル攻撃を受ける。 (*bô koku no tai kan mi sa i ru kô geki wo u ke ru*, « (ils) ont été attaqués par un missile sol-mer de tel pays »)
   ⇒ On ne trouve que « 対 » comme mot qui commence par le *kanji* 対. On considère alors la séquence de deux *kanji* « 対鑑 » comme un mot inconnu.

Par ailleurs, une autre méthode de traitement des mots inconnus plus évoluée est proposée par les créateurs du système MSLR dans l'article [25]. Bien que la version actuelle de MSLR ne dispose pas de ce mécanisme de traitement des mots inconnus, nous allons étudier dans la section suivante comment un système peut reconnaître les séquences non présentes dans le dictionnaire avec cette méthode.

5.3.6 Traitement des mots inconnus

Le programme de traitement des mots inconnus présume l'étendue du mot inconnu et sa catégorie à l'aide d'informations (provenant de l'analyseur)

antérieures à l'échec de l'analyse et d'informations statistiques des modèles vari-gram[13] obtenus préalablement à partir du corpus.

L'architecture du système avec l'unité de traitement des mots inconnus est présentée dans la figure 5.4.

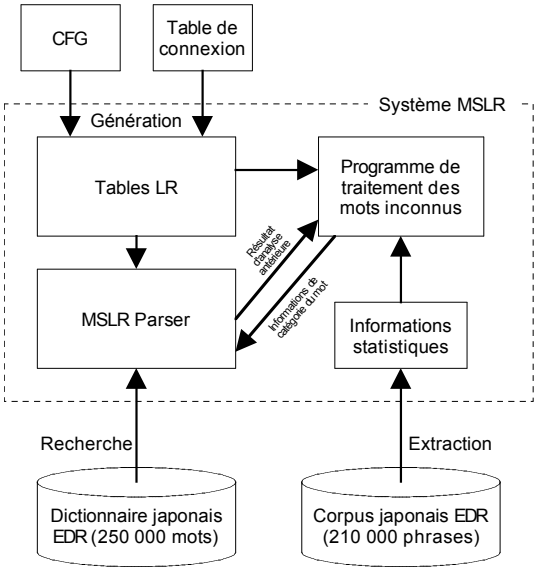


FIG. 5.4 – Architecture du système MSLR

La réalisation du traitement des mots inconnus suit les principes suivants :

1. Commencer le traitement lorsqu'on ne peut plus avancer l'analyse.
2. Pour présumer la catégorie du mot inconnu, on utilise d'une part la liste des catégories qui peuvent être traitées par les tables d'analyse LR, et d'autre part les informations des catégories obtenues par vari-gram.
3. Pour présumer l'étendue du mot inconnu, on utilise les informations de l'arbre syntaxique déjà généré, ainsi que les informations de l'étendue candidate dans laquelle la forme du texte entré correspond à un modèle de vari-gram.

Procédure de traitement

Nous allons voir à présent comment se réalise la procédure de traitement selon les principes décrits précédemment.

[13] À la différence des modèles n-gram (notamment bi-gram et tri-gram) qui traitent une longueur donnée n, les modèles vari-gram traitent des règles de connexion de longueur variant selon le besoin.

(1) Traitement des mots non présents dans le dictionnaire mais trouvés dans le corpus

Pour chaque état présent sur la pile construite avant l'échec de l'analyse, on crée une liste des catégories qui permettent une transition. Par exemple, si l'analyse a échoué avec l'action dans l'état i sur l'entrée N, on cherche dans les tables d'analyse LR les catégories permettant une transition à partir de l'état i. D'une façon plus concrète, on cherche sur la ligne de l'état i dans les tables LR les colonnes dans lesquelles une action est inscrite. Supposons que sur la ligne de l'état i, en plus de la colonne de substantif, les colonnes correspondant au qualificatif et au verbe soient remplies par une action `décaler`. On ajoute alors dans la liste des catégories le qualificatif et le verbe. Ce processus est représenté sur la figure 5.5.

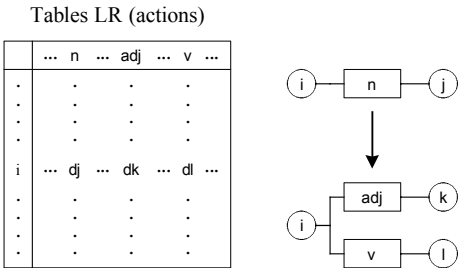


FIG. 5.5 – Présomption des catégories à partir des tables LR

On réalise ensuite une analyse morphologique à l'aide des modèles vari-gram en partant de la fin de phrase. Si la catégorie du morphème en tête appartient à la liste des catégories décrite précédemment, on conserve le résultat de cette analyse comme une possibilité. Après avoir trouvé toutes les possibilités, on calcule le coût de chaque possibilité et on prend comme résultat le plus probable la possibilité ayant le coût le moins élevé.

(2) Traitement des mots absents du dictionnaire et du corpus

Pour illustrer l'explication, utilisons l'exemple de l'article [25] avec la phrase :

交差点で事故ったらしい
(*kô sa ten de ji ko t ta ra shi i*)
« il paraît qu'il a eu un accident au carrefour »).

L'analyse de cette phrase échoue car le verbe « *ji ko ru* », construction encore familière à l'heure actuelle, ne figure pas dans le dictionnaire.

Au moment où l'analyse ne peut plus avancer, on revient en arrière de manière à déterminer la position initiale du mot inconnu. On cherche dans les données vari-gram les morphèmes correspondant à une partie de la séquence

entrée. Mais la séquence « *ji ko t ta* » n'étant pas dans le corpus, on ne peut pas trouver de candidat avec une correspondance totale.

On calcule alors le coût de toutes les hypothèses basées sur les modèles vari-gram[14] pour déterminer l'étendue du mot inconnu. Lors de la comparaison avec les modèles, on utilise comme information la forme des deux derniers morphèmes du modèle, la catégorie et la longueur du premier morphème. La catégorie du premier morphème est limitée dans ce cas également par la liste des catégories permettant une transition, comme expliqué précédemment. La longueur du morphème fournit l'information pour déterminer l'étendue du mot inconnu. Par exemple, avec le modèle 3-gram : « (qualificatif, longueur 1) いる。 », on trouve le candidat « し (qualificatif) いる (terminaison) 。 (symbole) ».

On calcule ensuite le coût pour chaque candidat à partir de la probabilité d'apparition dans le corpus. Par ailleurs, pour rendre le calcul plus efficace, on réalise d'abord l'élimination de certains candidats par une méthode heuristique. Par exemple, っ ne pouvant pas apparaître en tête de *jiritsugo* (mot autonome), les candidats commençant par le septième caractère du texte entré sont éliminés. D'après le résultat de calcul des coûts, le candidat « 事故 (vb, 2) っ (terminaison) た (auxiliaire) » est choisi comme résultat le plus probable.

Par cette méthode, le système est capable de présumer, lorsque le texte entré contient un mot qui ne figure ni dans le dictionnaire ni dans le corpus, l'étendue du mot inconnu et sa catégorie morphologique.

5.4 KNP

5.4.1 Introduction

Nous nous intéressons dans cette section au système d'analyse syntaxique du japonais KNP (Kurohashi-Nagao Parser), développé à l'Université de Kyoto.

La première caractéristique de cet analyseur se trouve dans sa capacité de détection des structures de coordination (syntagmes ou propositions). Il est caractérisé d'autre part par le fait qu'il n'utilise, à la différence des deux systèmes présentés dans les sections précédentes, aucun algorithme « classique » d'analyse syntaxique. En effet, comme le disent Kurohashi et Nagao dans [32] et [33] :

> « Les résultats d'une analyse syntaxique il n'y a pas si longtemps étaient "terribles". Mais, était-ce vraiment dû à des questions de sens ou de connaissances encyclopédiques ? Tel a été le point de départ des recherches et du développement du système d'analyse syntaxique du japonais KNP. [...] Nous avons considéré ces deux

[14] On considère ci-après que $n = 3$ pour simplifier l'explication.

> problèmes, c'est-à-dire l'analyse des structures de coordination et le traitement des expressions de type exceptionnel (dans le sens où on ne peut pas les traiter avec une grammaire simple et un nombre limité de productions) comme les problèmes à régler avant toute autre chose dans l'analyse syntaxique. »

Kurohashi et Nagao posent comme hypothèse que les constructions de coordination possèdant une certaine ressemblance, il est possible de traiter ces structures en détectant leur ressemblance. Cependant, selon eux « détecter leur ressemblance avec les méthodes traditionnelles ne peut pas être une solution ». Ils ont alors développé l'analyseur KNP, capable d'analyser ces structures de coordination, inspiré du principe de correspondance des méthodes de programmation dynamique, utilisées largement dans le traitement de la parole.

Par ailleurs, le système possède la possibilité de réaliser ou non une analyse de structure argumentale (*Case Frame*), appelée 格解析 (*kaku-kaiseki*, analyse des cas). Cependant, notre étude n'entre pas dans les détails de cette fonction.

La présente section est constituée de l'étude de la procédure générale d'analyse, du mécanisme d'analyse des structures de coordination et de l'analyse des relations de dépendance entre les syntagmes d'une phrase.

5.4.2 Procédure générale d'analyse

Schéma général

La figure 5.6 (voir page suivante) présente la procédure générale d'analyse par le système KNP. L'analyseur KNP utilise comme données le résultat d'une analyse morphologique réalisée par JUMAN [35], système d'analyse morphologique du japonais développé dans le même laboratoire de l'Université de Kyoto.

L'analyseur JUMAN réalise à l'aide du dictionnaire EDR [10] la segmentation des phrases en *tango* en s'appuyant sur les possibilités de connexion entre les unités et la reconnaissance de leur catégorie, *hinshi*, et de leur forme. L'analyseur KNP rassemble ces *tango*[15], issus de l'analyse par JUMAN, de manière à former des syntagmes *bunsetsu*.

KNP applique certaines opérations à ces unités résultant de l'analyse par JUMAN, de manière à obtenir ses unités de base, les syntagmes *bunsetsu*.

[15] Comme nous avons déjà fait la remarque dans la section 1.6.3 page 19, la plupart des unités traitées par JUMAN, définies comme *keitaiso* (morphèmes), ne sont pas toujours des unités morphologiques. Nous utilisons pour les unités segmentées par JUMAN le terme « *tango* », qui est souvent utilisé par les créateurs de KNP pour désigner les unités segmentés par JUMAN. Mais cet emploi ne correspond pas non plus à la définition de ce terme, car le préfixe *dai* de « *dai-seikô* » (grand + succès) est bien traité comme une unité.

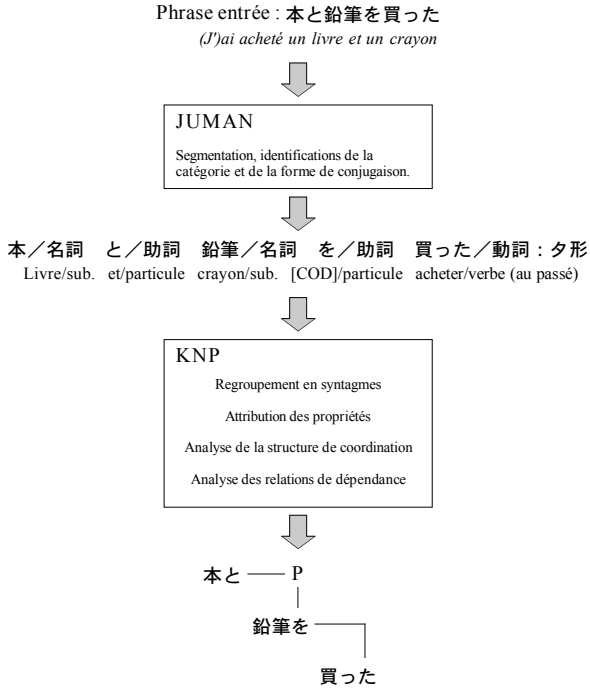


FIG. 5.6 – Analyse d'une phrase japonaise par JUMAN et KNP

Les opérations principales de KNP sont l'analyse des structures de coordination d'une part et l'analyse des relations de dépendance présentes entre les syntagmes, d'autre part. Ces opérations sont basées sur les propriétés attribuées à chaque *bunsetsu* constituant la phrase entrée. La procédure de traitement se déroule plus précisément de la façon suivante :

1. Traitement de l'homographie pour ne garder qu'une seule étiquette par mot.
2. Attribution aux *tango* de propriétés concernant leur sens et concernant la catégorie servant à constituer les syntagmes *bunsetsu*.
3. Constitution des syntagmes *bunsetsu* selon les propriétés attribuées.
4. Attribution aux *bunsetsu* de propriétés telles que leur fonction syntaxique (cas de *wo*, cas de *ga*, etc.) et le marqueur de la structure de coordination.
5. Détection des syntagmes ayant un caractère commun, dans le cas où il existe une expression pouvant être un marqueur de structures de coordination.
6. Analyse et construction des relations de dépendance dans l'ensemble de la phrase (sans trahir la structure de coordination détectée).

7. En mode sans analyse de la structure argumentale : choix et sortie du résultat le plus probable par évaluation de toutes les possibilités obtenues.
8. En mode avec analyse de la structure argumentale : recherche dans le dictionnaire de structures argumentales (*Case frame*), de toutes les combinaisons « prédicat – arguments » obtenues, pour déterminer le cas profond de chaque argument, puis évaluation de toutes les structures de dépendance, en retenant comme critère de choix le degré de correspondance avec le dictionnaire, afin de sortir le résultat le plus probable et les informations sur ses cas profonds.

Nous verrons dans la section 5.4.3 les quatre premières opérations préparatoires décrites précédemment, puis nous passerons ensuite dans la section 5.4.4 page 135 à l'étude de l'analyse des structures de coordination (5.) et de celle des relations de dépendance (6.), opérations principales du système.

5.4.3 Opérations préparatoires

Définition de l'unité de base, *bunsetsu*

Avant de commencer, nous présentons la définition de KNP de la notion de syntagme *bunsetsu*, unité de base pour l'analyse des relations de dépendance, donc unité de base pour ce système d'analyse syntaxique.

Définition 25 (Syntagme *bunsetsu*)
Nous appelons bunsetsu l'unité constituée d'au moins un jiritsugo (mot autonome, tel que substantif, verbe) et de zéro, un ou plusieurs fuzokugo (mot attaché, tel que particule, suffixe).

Traitement de l'homographie

Ambiguïté dans les résultats de l'analyse morphologique L'analyse morphologique, qui réalise la segmentation en *tango* et l'attribution de catégories à ces unités, entraîne en général des résultats multiples. Cette ambiguïté s'étend sur deux niveaux.

Premièrement, la multiplicité des résultats peut se trouver sur le plan de la segmentation. Pour ces données multiples, le système KNP choisit parmi ces possibilités le résultat constitué du moins d'unités et du moins de mots autonomes. Ce choix est expliqué dans [33] par le fait que le système s'intéressant surtout à la détection des structures de coordination, il est plus intéressant de traiter les unités composées comme une seule unité.

Deuxièmement, une séquence peut correspondre à plusieurs types de *tango*. Le système traite ces homographes en choisissant un seul *tango* selon des règles définies. Avant de regarder quelques exemples de règles, nous allons introduire la représentation des *tango*, basée sur celle de JUMAN.

Définition 26 (Représentation des *tango*)
Un tango est représenté sous la forme d'un quintuple, appelé structure de tango :

[Cat SCat TypConj FrmConj Frm]

où
- *Cat représente le nom de la catégorie, hinshi,*
- *SCat représente le nom de la sous-catégorie, hinshi-saibunrui,*
- *TypConj représente le nom du type de conjugaison,*
- *FrmConj représente le nom de la forme de conjugaison,*
- *Frm représente la forme superficielle du tango.*[16]

Les règles sont décrites sous forme de
(($tango_1$ $tango_2$ ... $tango_n$) série de propriétés)
Si la séquence contient le même ensemble de *tango*, le premier *tango* ($tango_1$) dans la règle doit être choisi. Ce fichier contient 65 règles de traitement de l'homonymie, classées dans quatre catégories :

1. règles des mots non définis, traitement des homographes représentant un conflit entre les mots non définis et différents type de substantifs ;
2. règles des substantifs, traitement des homographes représentant un conflit entre différents types de substantifs ;
3. règles des autres catégories uniques, traitement des homographes représentant un conflit entre différentes sous-catégories de même catégorie autre que les mots non définis et les substantifs ;
4. règles de catégories différentes, traitement des homographes représentant un conflit entre différentes catégories, tel qu'un conflit verbe – substantif.

L'ordre des règles est pertinent car l'application des règles se réalise suivant leur ordre d'apparition et l'opération s'arrête une fois qu'une règle a été appliquée.

Nous allons étudier quelques règles pour illustrer l'explication ci-dessus. Par exemple, la règle du premier type (mots non-définis) :

(([名詞　組織名] [未定義語　アルファベット]))
(([substantif nom-d'organisation] [mot-non-défini alphabet]))

signifie que dans le cas des homographes pouvant être un *tango* de la sous-catégorie 組織名 (*soshikimei*, nom d'organisation) de la catégorie 名詞 (*meishi*, substantif), ou un *tango* de la sous-catégorie アルファベット (*arufabetto*,

[16] Ceci est une traduction fidèle de la définition du manuel, mais il s'agit, d'après les utilisations concrètes observées dans différents fichiers, d'une forme lématisée, c'est-à-dire une forme canonique de type « entrée de dictionnaire ».

alphabet) de la catégorie 未定義語 (*miteigigo*, mot non défini), on choisit l'étiquette de tête, soit « substantif - nom d'organisation ». Par exemple, si nous considérons une séquence telle que « NTT », qui possède les deux étiquettes, l'application de la règle ne retient qu'une seule étiquette, la première : « substantif – nom d'organisation ».

Des séquences telles que 引き上げ (*hikiage*, augmentation), 伸び (*nobi*, croissance), 仕掛け (*shikake*, mécanisme), provoquant un conflit verbe – substantif, sont considérées comme un verbe ayant un caractère substantif d'après la règle du quatrième type (catégories différentes) :

(([動詞] [名詞]) 名詞的動詞)
(([verbe] [substantif]) verbe ayant un caractère substantif)

Si le *tango* suivant avec lequel il constitue un syntagme est une particule, ce syntagme prend l'étiquette « nominal », mais si cette séquence constitue toute seule un syntagme, l'étiquette est « verbal ».

Attribution de propriétés aux *tango*

La représentation des *tango* dans KNP est identique à la représentation des *tango* de JUMAN, dans laquelle leurs propriétés sont ajoutées à la fin. Ainsi, un *tango* est représenté sous forme de sextuple :

[Cat SCat TypConj FrmConj Frm Proriétés].

Ce dernier et nouvel élément, ensemble des propriétés du *tango* telles que son sens ou la catégorie servant à constituer les syntagmes *bunsetsu*, est attribué selon des règles définies et décrites sous la forme de :

((MP) (MT) (MS) série de propriétés)

où MP représente les *tango* précédents, MT le *tango* à traiter, et MS les *tango* suivants. L'ensemble des propriétés du dernier élément est attribué au *tango* à traiter se situant au milieu.

Environ 150 règles sont définies. Le traitement se déroule à partir du *tango* du début de la phrase. L'application des règles se réalise selon leur ordre d'apparition et ce sans interruption même si une règle a été trouvée. L'ordre des règles est toutefois pertinent car certaines utilisent des propriétés attribuées par des règles précédentes.
Les règle sont classées dans huit catégories :

1. règles des mots non définis ;
2. règles des informations de type dictionnaire ;
3. règles des verbe de type *fuzokugo* ;
4. règles des symboles ;
5. règles des modifications de catégorie ;

6. règles des suffixes transformant le type *yôgen* en *taigen* et inversement ;
7. règles des préfixes et des *fuzokugo* ;
8. règles des mots composés.

Les règles des mots non définis attribuent aux mots non définis une catégorie selon le type de caractère (*katakana*, alphabet ou autre) et la propriété « ambiguïté de catégorie ».

Les règles des informations de type dictionnaire attribuent les propriétés telles que « mot de coordination », « spécifique numéral de temps », ou encore « candidat de verbe de type *fuzokugo* », etc. Par exemple, la règle :

```
( ( ) ( [* * * * (および　及び　または　又は　また　ならびに あるいは　或いは　もしくは　若しくは　そして　ないしは　ないし かつ　かつまた　それとも) ] ) ( ) 一般並列語 )
```

signifie qu'aux *tango* および／及び (*oyobi*, et), または／又は (*matawa*, ou), また (*mata*, ou), ならびに (*narabini*, et), あるいは／或いは (*aruiwa*, ou), もしくは／若しくは (*moshikuwa*, ou), そして (*soshite*, et), ないしは (*naishiwa*, ou), ないし (*naishi*, ou), かつ (*katsu*, et aussi), かつまた (*katsumata*, et aussi) et それとも (*soretomo*, ou), on attribue la propriété « 一般並列語 (*ippanheiretsugo*, mot de coordination) ».

Les règles des verbes de type *fuzokugo* attribuent la propriété « *fuzokugo* » aux *tango* ayant comme propriété « candidat de verbe de type *fuzokugo* » et précédés par certains types de *tango*, au qualificatif 欲しい (*hoshii*, désireux) précédé par certains types de *tango*, ainsi qu'au verbe 下さる (*kudasaru*, avoir la bonté de) précédé par le préfixe お (*o*, [embellissement]) suivi d'un verbe.

Les règles des symboles attribuent aux *tango* constitués de symboles, de lettres alphabétiques, de chiffres ou de *katakana*, la propriété « 記英数カ (symbole-alphabet-chiffre-*katakana*) » et aux *tango* en *kanji* la propriété « 漢字 » (*kanji*).

Les règles des modifications de catégorie définissent les conditions de transformation de catégorie. Par exemple, la règle :

```
( ( ) ( [動詞 * * (基本連用形] ) ( [動詞 * サ変動詞 * する] )
                &品詞変更：名詞：サ変名詞 )
```

```
( () ([verb * * (forme renyô de type basique]) ([verb * verbe sahen * faire])
        & changement de catégorie : substantif : substantif de type sahen )
```

signifie qu'aux verbes à la forme *renyô* de type basique suivi du verbe de type *sahen* する (*suru*, faire), on attribue la propriété « modification de catégorie en substantif de type *sahen* ».

Les règles des suffixes transformant le type *yôgen* en *taigen* et inversement attribuent la propriété « transformation *yôgen* – *taigen* » aux suffixes de type nominal succédant à un prédicat, aux suffixes de type qualificatif succédant à un substantif et au suffixe de type verbal めく (*meku*, paraître).

Les règles des préfixes et des *fuzokugo* attribuent trois types de propriétés : 自立(*jiritsu*), 付属(*fuzoku*) et 接頭 (*settô*). La première signifie que l'unité est un mot autonome *jiritsugo*, la deuxième signifie qu'elle est un mot annexe *fuzokugo*, et la troisième signifie qu'il s'agit d'un préfixe *settôgo*. Étant donnée l'importance de ces notions, à la base même de la constitution des syntagmes *bunsetsu*, étudions de plus près ces règles.

On distingue cinq types de règles :

1. les règles par défaut ;
2. les règles concernant l'unité de type *sahen*[17] ;
3. la règle sur les substantifs formels[18], 形式名詞 (*keishiki-meishi*) ;
4. les règles sur les locutions ;
5. les autres règles.

Avec les règles par défaut, les mots sont classés comme suit :

- *fuzokugo* : les *jodôshi*, les suffixes, les mots de jugement, les particules, les mots de catégorie spéciale tels que les signes de ponctuation, la parenthèse fermante, le signe égal, le double tiret, les points de suspension ;
- préfixes : les préfixes et la parenthèse ouvrante.

Avec les règles concernant l'unité de type *sahen*, les mots suivants sont classés comme dans les *fuzokugo* :

- les verbes する (*suru*, faire), できる／出来る (*dekiru*, être capable), いたす／致す (*itasu*, faire) lorsqu'ils suivent une unité de type *sahen* éventuellement suivie d'une parenthèse fermante ;
- les qualificatifs 可能だ (*kanôda*, possible), 不可能だ (*fukanôda*, impossible) lorsqu'ils suivent une unité de type *sahen* éventuellement suivie d'une parenthèse fermante ;
- les unités 前 (*mae*, avant), 中 (*chû*, pendant), 後 (*go* / *ato*, après), 以前 (*izen*, avant), 以後 (*igo*, après), 以来 (*irai*, depuis), 以降 (*ikô*, après), 直前 (*chokuzen*, juste avant), 途中 (*tochû*, au milieu de), 直後 (*chokugo*, juste après), 予定 (*yotei*, projet) lorsqu'elles suivent une unité de type *sahen* éventuellement suivie d'une parenthèse fermante ;

[17] Ce sont des substantifs qui peuvent constituer un verbe – dit verbe *sahen* – lorsqu'ils sont suivis par le verbe する (*suru*).

[18] Il s'agit de substantifs ayant perdu leur sens et qui servent essentiellement à regrouper les syntagmes qu'ils précèdent pour nominaliser ce bloc.

- le verbe する (*suru*, faire) lorsqu'il suit la séquence « verbe à la forme volitive + particule *to* » pour former le locution しようとする (*shiyô-tosuru*, essayer de faire).

La règle sur les *keishiki-meishi* définit que le substantif formel *no* est toujours *fuzokugo*.

Les règles sur les locutions définissent que :

- dans certaines locutions le verbe est considéré comme *fuzokugo* ;
- dans la locution かどうか（も）(*kadôka(mo)*, [on ne sait pas] si) le démonstratif どう (*dô*, comment) est considéré comme *fuzokugo* ;
- les adverbes ともに／共に (*tomoni*, avec), どうじに／同時に (*dô-jini*, en même temps) suivis de la particule *to* sont considérés comme *fuzokugo*.

Les autres règles sont constituées de six règles :

- les unités considérées comme des mots de coordination suivis d'une virgule, sont des *fuzokugo* ;
- le mot その他 (*sonota*, autre) suivi d'une virgule est un *fuzokugo* ;
- le mot 等 (*tô / nado*, etc.) suivi d'une virgule est un *fuzokugo* ;
- les substantifs 以後 (*igo*, après), 以来 (*irai*, depuis), 以降 (*ikô*, après) suivant un verbe à la forme en *te*, sont des *fuzokugo* ;
- le mot 自体 (*jitai*, soi-même) est un *fuzokugo* ;
- toute autre unité, qui n'est ni *fuzokugo* ni préfixe, est un mot autonome.

Les règles des mots composés attribuent au *tango* traité remplissant certaines conditions quatre types de propriétés :

1. mot qui peut constituer un mot composé ;
2. mot qui, lorsqu'il constitue un mot composé, peut se composer avec une unité située à sa gauche ;
3. mot qui, lorsqu'il constitue un mot composé, peut se composer avec une unité située à sa droite ;
4. mot qui se compose (dans la phrase analysée) avec l'unité précédente.

Par exemple, la règle :

(( ) ([副詞 * * * * ((漢字 記英数カ))]) ( )
T→複合？ T複合？→ T複合？)

(() ([adverbe * * * * ((*kanji* symbole-alphabet-chiffre-*katakana*))]) ()
T→Composé？ T Composé？→ T Composé？)

définit que les adverbes 副詞 (*fukushi*) écrits en *kanji*, symbole, lettre, chiffre ou katakana (漢字 記英数カ), peuvent se lier avec une unité qui les suit ou qui les précède. Ainsi, les mots composés tels que 突然死 (*totsuzen / shi*, soudain / mort, « mort subite »), 直接選挙 (*chokusetsu / senkyo*, directement / élection, « suffrage direct »), peuvent être correctement analysés. Pour

empêcher d'appliquer cette règle à une séquence telle que 普通われわれは (*futsû / wareware wa*, en général / nous [thème] « Nous (faisons) en général », est définie la règle :

(( ) ([副詞]) ([(名詞　普通名詞 * * * ((¬漢字 ¬記英数カ))])
¬T複合→)

((()([adverbe])([(substantif nom-commun * * * ((¬kanji ¬symb-alph-chffr-kata))])
¬TComposé ?→)

qui enlève aux adverbes 副詞 la possibilité de se lier avec un substantif qui les suit, s'il est de type nom commun 名詞　普通名詞 non écrit en *kanji*, symbole, alphabet, chiffre ou *katakana* (¬漢字 ¬記英数カ).

Reconnaissance des syntagmes *bunsetsu*

Regroupement des *tango* en syntagmes *bunsetsu* KNP regroupe ensuite, selon les propriétés attribuées à chaque *tango*, un ou plusieurs *jiritsugo* avec zéro, un ou plusieurs *fuzokugo* qui les succède(nt) de manière à former des syntagmes *bunsetsu*.

Les règles d'attribution des propriétés sont définies de telle manière qu'à cette étape le système KNP crée un certain nombre de syntagmes « exceptionnels » qui ne répondent pas à la définition de *bunsetsu*, mais qu'il est plus intéressant de considérer comme des *bunsetsu*. En fait, c'est un type d'expression exceptionnelle, considéré par les créateurs de KNP comme la source d'un des deux grands problèmes à régler avant tout dans l'analyse syntaxique, ce que nous avions évoqué dans l'introduction de cette section. Pour le traitement de ces expressions, Kurohashi explique dans [32] : « si on ne les traite pas correctement, cela entraîne un résultat d'analyse complètement erroné. Mais étant donné que ce sont des problèmes localisés à chacune de ces expressions, il suffit de définir correctement leur fonction. [...] Le problème est comment on peut collecter ces expressions ».
Les exemples de ces expressions citées sont les suivants :

1. 書き損じる (*kaki sonjiru*, mal écrire) ;
2. 読み飛ばす (*yomi tobasu*, omettre de lire) ;
3. Verbes à la forme *mizen* + ざるをえない (*zaruwoenai*, être obligé de faire q.c.) ;
4. Verbes en *ta* + にちがいない (*nichigainai*, il est certain que) ;
5. Verbes à la forme *shûshi* + とはいえ (*towaie*, bien que) ;
6. Verbes à la forme *shûshi* + のかどうかさえ (*nokadoukasae*, ne même pas (savoir) si) ;
7. Verbes à la forme *mizen* + ずじまい (*zujimai*, finalement ne pas faire q.c.) ;

8. Verbes à la forme d'intention + ものなら (*mononara*, si q.c. se réalisait) ;
9. Verbes en *ta* + 途端 (*totan*, dès) ;
10. Verbes en *ta* + 矢先 (*yasaki*, juste au moment de) ;
11. Verbes en *te* + 以降 (*ikô*, depuis) ;
12. Verbes en *te* + 以来 (*irai*, depuis) ;
13. Verbes en *te* + 以後 (*igo*, après) ;
14. Verbes à la forme *renyô* + 次第 (*shidai*, dès).

Par ailleurs, les expressions avec un substantif de type *sahen* telles que 「サ変名詞＋する」 (sub. *sahen* + *suru* (faire)) sont considérées comme un *bunsetsu*. Dans ce cas, on considère le verbe comme un *fuzokugo*, et son *jiritsugo* est considéré comme un verbe.

De plus, les conjonctions telles que または (*matawa*, ou), あるいは (*aruiwa*, ou) suivant une virgule, sont considérées comme *fuzokugo*, formant ainsi un *bunsetsu* avec un ou des *jiritsugo* précédant la virgule.

Attribution de propriétés aux syntagmes *bunsetsu*

Cette étape d'attribution de propriétés aux *bunsetsu* représente la colonne vertébrale de cet analyseur.
Les syntagmes *bunsetsu* sont représentés :

((séquence de *tango*) propriétés)

Par exemple[19],

((?* [助詞 * * * と] [助詞 * * * も]* [特殊 読点]*) ((用言)))
((?* [particule * * * *to*] [particule * * * *mo*]* [spécial virgule]*) ((*yôgen*)))

représente les syntagmes constitués de séquences telles que 「と」「と、」「とも」「とも、」 suivies de séquences quelconques, ayant comme propriété 用言 (*yôgen*).

Le dernier élément, ensemble des propriétés du syntagme, est attribué selon les règles définies et regroupées dans cinq fichiers différents, et décrites sous forme de :

((SP) (ST) (SS) série de propriétés),

où SP représente les syntagmes précédents, ST le syntagme à traiter, et SS les syntagmes suivants. L'ensemble des propriétés du dernier élément est attribué au syntagme à traiter se situant au milieu.
Le traitement se déroule à partir du syntagme de la fin de la phrase. L'application de règles se réalise selon leur ordre d'apparition. L'ordre est ici aussi pertinent pour ces règles.

Étudions à présent les règles par fichier.

[19] ? désigne un *tango* quelconque et * suivi d'une représentation de *tango* signifie la répétition, supérieure ou égale à zéro fois, de ce *tango*.

Le fichier `bnst_basic.rule` regroupe les règles attribuant différentes propriétés principales, telles que :

- propriétés transmises par le mot autonome qui constitue le syntagme, « qualification de quantité » par exemple ;
- propriétés 地名(*chimei*, nom de lieu), 人名(*jinmei*, nom de personne), 組織名(*soshikimei*, nom d'organisme) qui signifient que le syntagme contient un mot, qui est un nom de lieu, un nom de personne ou un nom d'organisme. Ces propriétés servent à calculer la ressemblance entre les syntagmes ;
- propriétés 名並終点(fin de coordination des substantifs), 述並終点(fin de coordination des prédicats) qui marquent la limite de la structure de coordination ;
- propriétés marquant la présence de mots exprimant certaines voix comme させる (suffixe formant le factitif)[20]. Ces propriétés servent à maîtriser la modification de la structure argumentale due au changement de voix ;
- propriétés marquant la particule qui regroupe le syntagme.

L'application de ces règles s'arrête dès qu'une règle a été trouvée.

Le fichier `bnst_etc.rule` regroupe les règles attribuant une propriété exceptionnelle, telle que celles qui peuvent servir à maîtriser la portée de la particule *wa*. L'application de ces règles ne s'arrête pas même si une règle a été trouvée.

Le fichier `bnst_uke.rule` regroupe les règles attribuant une propriété de réception, telles que :

- 体言 : qui est attribué aux syntagmes nominaux ;
- 用言 : 強 ; 動 : qui est attribué aux syntagmes verbaux. Cette propriété de réception est considérée comme « forte » ;
- 用言 : 強 ; 形 : qui est attribué aux syntagmes constitués d'un qualificatif. Cette propriété de réception est considérée comme « forte » ;
- 用言 : 強 ; 判 : qui est attribué aux syntagmes constitués d'un mot de jugement. Cette propriété de réception est considérée comme « forte » ;
- 用言 : 弱 ; 隣 D : qui est attribué aux syntagmes pour lesquels seul le syntagme juste à leur gauche peut être leur complément. Cette propriété de réception est qualifiée de « faible » ;
- 用言 : 弱 ; 隣 d : qui est attribué aux syntagmes pour lesquels seul le syntagme juste à leur gauche peut être leur complément à condition qu'à leur droite ne se trouve pas un vrai *yôgen*. Cette propriété de réception est qualifiée de « faible » ;

[20] La voix est définie comme une catégorie grammaticale qui indique la relation entre le verbe, l'agent et l'objet. En japonais, il existe les voix passive, passive indirecte, factitive, potentielle et la voix dite 自発 (*jihatsu*, changement automatique)

- 用言：弱 : qui est attribué aux syntagmes ne prenant aucun complément. Cette propriété de réception est qualifiée de « faible » ;
- 受：感動詞, 受：副詞形態指示詞, 受：連体詞形態指示詞, 受：接続詞 : qui est attribué aux syntagmes constitués d'un verbe ou d'un qualificatif à la forme *renyô*.

L'application de ces règles s'arrête dès qu'une règle a été trouvée. Tout syntagme est supposé être concerné par une règle. Il existe une dernière règle qui attribue aux syntagmes la propriété « aucune propriété de réception », qui sert en fait à signaler l'absence de règle correspondante.

Le fichier `bnst_uke_ex.rule` regroupe les règles attribuant des propriétés exceptionnelles de réception, concernant principalement les locutions. L'application de ces règles ne s'arrête pas même si une règle a été trouvée.

Le fichier `bnst_kakari.rule` regroupe les règles attribuant une propriété de dépendance, telle que :

- 係：連用 : qui est attribué aux syntagmes constitués d'un verbe ou d'un qualificatif à la forme *renyô* ;
- 係：連体 : qui est attribué aux syntagmes constitués d'un verbe ou d'un qualificatif à la forme *rentai* ;
- 係：ガ格 : qui est attribué aux syntagmes regroupés par la particule *ga* ;
- 係：ヲ格 : qui est attribué aux syntagmes regroupés par la particule *wo* ;
- 提題 : qui est attribué aux syntagmes regroupés par la particule marquant le thème ;
- 係：文末 : qui est attribué aux syntagmes finissant par un point (fin de citation entre parenthèses).

L'application de ces règles s'arrête lorsqu'une règle a été trouvée. Tout syntagme est supposé être concerné par une règle. Il existe une dernière règle qui attribue aux syntagmes la propriété « aucune propriété de dépendance », qui sert, comme pour la propriété de réception, à signaler l'absence de règle correspondante.

Les règles figurant dans l'ensemble de ces fichiers attribuent également aux syntagmes différentes informations : le niveau de dépendance, les informations sur les expressions de coordination, l'identité de la règle appliquée et le niveau de limitation des structures de coordination – notion que nous allons maintenant étudier.
Considérons d'abord la phrase :

この　装置　は、生産　と　検査　の　自動化　を　実現する
(ce - équipement - [thème] - production - et - contrôle - de - automatisation - [COD] - réaliser)
« Cet équipement réalise l'automatisation de la production et du contrôle. »

Il est difficile de considérer que le syntagme この装置は (cet équipement [thème]) appartient à la structure de coordination marquée par la particule と (et). En tenant compte des phénomènes de ce type, on définit les éléments pouvant marquer une sorte de limite tels que la particule は ([thème]) dont la portée est généralement très large. Afin de pouvoir utiliser ensuite ce constat pour le calcul des structures de coordination, les éléments marquant la limite sont classés en niveaux de 1 à 5 selon leur capacité de limitation.

5.4.4 Reconnaissance des structures de coordination

Nous étudions à présent la première opération principale du système KNP, la détection des structures de coordination, qui est réalisée avant l'analyse des relations de dépendance, seconde opération principale de ce système que nous étudierons dans la section 5.4.5 page 143. Cette étude s'appuie essentiellement sur l'étude de [33] et [34].

Le système détecte, à l'aide des propriétés attribuées pendant la procédure préparatoire décrite précédemment, les structures de coordination en examinant les syntagmes de manière à leur trouver un caractère commun.

Avant d'entrer dans les détails du mécanisme de détection, nous allons nous intéresser à la définition des structures de coordination pour le système KNP.

Types de structure de coordination

1) Coordination des substantifs :

解析　と　生成　を　行う
(analyse - et - génération - [COD] - réaliser)
« (Il) réalise l'analyse et la génération »

Il s'agit d'une structure dans laquelle deux substantifs sont coordonnés avec un coordonnant.

2) Coordination des prédicats :

原言語　を　解析し　相手言語　を　生成する
(langue source - [COD] - analyser - langue cible - [COD] - générer)
« (Il) analyse la langue source et génère la langue cible »

解析　では　利用する　が、生成　では　利用しない
(analyse - dans - utiliser - mais - génération - dans - utiliser (négation))
« (Il l') utilise dans l'analyse mais (il ne l') utilise pas dans la génération »

Il s'agit d'une structure ayant deux prédicats de même statut, c'est-à-dire qu'ils n'ont pas de relation gouvernant / subordonné. Ces prédicats peuvent être reliés en mettant celui de la première proposition à la forme connective (comme dans le premier exemple), ou par une particule de conjonction (comme dans le second exemple).

3) Coordination partielle :

前者　を　解析　に、後者　を　生成　に　利用する
(le premier - [COD] - analyse - pour - le second - [COD] - génération - pour - utiliser)
« (On) utilise le premier pour l'analyse et le second pour la génération »

解析　に、または　生成　に　利用する
(analyse - pour - ou - génération - pour - utiliser)
« (On l') utilise pour l'analyse ou pour la génération »

Il s'agit de la coordination d'une certaine partie de deux propositions à l'exception de leur prédicat[21]. Pour ce type de coordination, il existe des cas où l'on constate ce statut coordonné par la présence d'une (de) même(s) particule(s) (comme dans le premier exemple), et d'autres où la coordination peut être réalisée par l'utilisation d'un coordonnant (comme dans le second exemple).

Coordonnants

La coordination peut être explicite. Dans ce cas, les deux éléments coordonnées sont reliés par un coordonnant. Les créateurs du système citent dans [33] comme coordonnants les mots ou les expressions suivants, en les catégorisant selon le type de structure de coordination où ils apparaissent :

(a) pour la coordination des substantifs : point virgule, point centré, と (et), も (aussi), や (et), か (ou), とか (et), かつ (et aussi), だけで（は）なく (non seulement), および (ou), または (ou), ならびに (et), あるいは (ou), もしくは (ou) ;

(b) pour la coordination des prédicats : のに対し（て） (tandis que), とか (et), か (ou), し (et), が (mais), ずに (sans), だけで（は）なく (non seulement), けれど（も） (mais), および (et), または (ou), ならびに (et), あるいは (ou), もしくは (ou) ;

(c) pour la coordination partielle : および (et), または (ou), ならびに (et), あるいは (ou), もしくは (ou).

Le système les considère comme des repères pour la reconnaissance des structures de coordination en y ajoutant d'autres formes susceptibles de constituer ce type de structure de coordination, telles que la forme connective de *yôgen*, la répétition d'une même particule, ou la présence du substantif *sahen*, etc. Lorsque le système trouve un de ces repères dans la phrase à traiter, il cherche à déterminer l'étendue des deux syntagmes constituant la structure de coordination.

[21] Il s'agit d'une traduction fidèle de la définition décrite dans [33]. Il me semble plus correct de reformuler cette définition, par exemple en « coordination de deux structures, appartenant au même paradigme ».

Méthode de détection des structures de coordination

La détection des syntagmes coordonnés est constituée des étapes suivantes :

1. repérage des clés de coordination ;
2. calcul du degré de ressemblance des syntagmes ;
3. présomption de l'étendu de la structure de coordination.

Repérage des clés de coordination Cette étape consiste à trouver les syntagmes satisfaisant une certaine condition et à les considérer comme clés de la coordination.

- Pour la coordination des substantifs, on considère comme clé le syntagme ayant un coordonnant et dont le mot autonome est un substantif.
- Pour la coordination des prédicats, on cherche d'abord le syntagme dont le mot autonome est un verbe ou un mot de jugement. Lorsqu'il y a un syntagme correspondant, on le considère comme clé s'il contient un coordonnant ou si la forme de conjugaison du prédicat[22] est *renyô* et que son *fuzokugo* contient une virgule.
- Pour la coordination partielle, on cherche d'abord le syntagme dont le mot autonome est un substantif et dont le fuzokugo contient une particule qui n'est pas considérée comme coordonnant de la coordination des substantifs et une virgule. Lorsqu'il y a un syntagme correspondant, on le considère comme clé, soit si le couple constitué de la particule qu'il contient et d'une autre particule (sauf *no*) d'un syntagme antérieur plus proche apparaît également dans la partie postérieure de la phrase, soit si le coordonnant de la coordination partielle apparaît après la virgule comme *fuzokugo*.

Calcul du degré de ressemblance des syntagmes Cette étape consiste en un calcul du degré de ressemblance de tous les couplages possibles de tous les syntagmes.
L'algorithme de calcul est comme suit :

Données : S_1 et S_2, deux syntagmes de la phrase entrée
Résultat : $f(S_1, S_2)$, degré de ressemblance des deux syntagmes

[1] **Si** les catégories du *jiritsugo* de S_1 et de S_2 sont identiques, **alors**
   (a) incrémenter $f(S_1, S_2)$ de 2 points ;
   (b) **Si** les *jiritsugo* de S_1 et de S_2 sont identiques, **alors**

[22] La forme de conjugaison du syntagme est celle du mot autonome quand le syntagme ne contient aucun *fuzokugo* variable, et celle du dernier *fuzokugo* s'il contient un ou plusieurs *fuzokugo* variables.

i. incrémenter $f(S_1, S_2)$ de 10 points ;

(c) **sinon**

i. **Si** les *jiritsugo* de S_1 et de S_2 sont des substantifs,
alors incrémenter $f(S_1, S_2)$ de $pt = 2 \times n$ points, où n est le nombre de caractères communs de S_1 et de S_2, et $pt \leq 10$;

ii. $cd :=$ les codes du *jiritsugo* de S_1 et de S_2, définis dans la Table des lexiques catégorisés[23] ;

iii. **Si** les cd de S_1 et de S_2 contiennent à partir de la gauche une même séquence de chiffres de longueur ≥ 3,
alors incrémenter $f(S_1, S_2)$ de $cdpt = 2 \times n$ points, où $n =$ (nombre de chiffres communs à S_1 et S_2) $- 2$, et $pt + cdpt \leq 10$ ($cdpt = 0$, si au moins un des deux *jiritsugo* ne figure pas dans cette table) ;

(d) incrémenter $f(S_1, S_2)$ de $fpt = 3 \times n$ points, où n est le nombre de *fuzokugo* communs de S_1 et de S_2 ;

[2] **Sinon**

(a) **Si** S_1 ou S_2 contient un substantif *sahen* considéré comme substantif et l'autre un substantif *sahen* considéré comme verbe,
alors incrémenter $f(S_1, S_2)$ de 2 points et continuer le calcul des points en appliquant [1](b) et [1](c) ;

(b) **Si** S_1 ou S_2 est une clé de la coordination des prédicats et que l'autre est un syntagme susceptible d'être un prédicat[24], situé dans une partie de la phrase postérieure au premier syntagme,
alors incrémenter $f(S_1, S_2)$ de 2 points ;

L'opération [1] correspond au calcul des structures de coordination des unités de même catégorie. L'opération [2] traite les structures de coordination où deux unités de catégories différentes sont coordonnées. Par exemple,

解析、生成する
(analyser (omission de partie variable) - générer)
« (Il) analyse et génère »

頑強で、分野　を　問わない　システム
(robuste - domaine - [COD] - choisir (négation) - système)
« Un système robuste utilisable dans tous les domaines »

Le premier exemple, phénomène localisé aux verbes *sahen*, concerne [2] (a), et le second exemple dans lequel le qualificatif de forme *renyô* et le verbe sont coordonnés, concerne [2] (b).

[23] Cette table est produite par le 国立国語研究所, *The national language research institute*

[24] Un syntagme peut être un prédicat, soit si son *jiritsugo* est un *yôgen*, soit si un mot de jugement appartient à ses *fuzokugo*.

Présomption de l'étendu de la structure de coordination Dans les parties antérieure et postérieure au syntagme clé de coordination, on cherche un couple de séquences de syntagmes pour lequel la somme de points de ressemblance est la plus élevée. Ce couple est calculé de la manière décrite ci-dessous à l'aide d'une matrice triangulaire, comme représenté sur la figure 5.7, constituée d'éléments (i, j) représentant les points de ressemblance du $i^{\text{ème}}$ syntagme et du $j^{\text{ème}}$ syntagme.

Définition 27 (Matrice partielle)
On appelle matrice partielle la partie rectangulaire en haut à droite de la ligne du syntagme clé de coordination.

Dans la figure 5.7, « a> » marque le syntagme clé de coordination et la partie encadrée d'une ligne discontinue correspond à la matrice partielle.

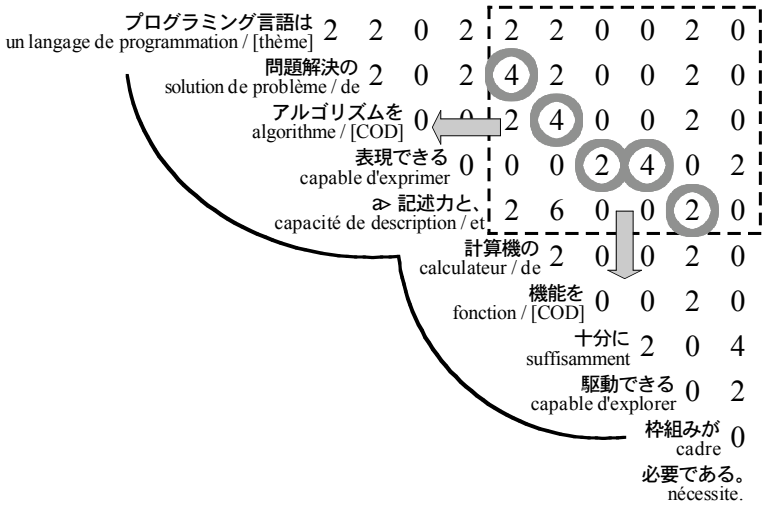


FIG. 5.7 – Analyse d'une structure de coordination basée sur la ressemblance

Définition 28 (Chemin)
On appelle chemin le parcours partant d'un élément non-nul de la ligne du syntagme clé, dit point de départ, et se terminant sur un élément de la colonne la plus à gauche de la matrice partielle.

La première opération est le calcul du score de tous les chemins pour chaque élément non-nul de la ligne du syntagme clé, selon la méthode que nous allons étudier plus bas.

Définition 29 (Meilleur chemin)
On appelle meilleur chemin le chemin ayant le meilleur score parmi les chemins partant d'un même élément.

Parmi les meilleurs chemins obtenus pour chaque point de départ correspondant à l'élément de la ligne du syntagme clé, le chemin ayant le meilleur score indique l'étendue de la structure de coordination.

Dans une phrase japonaise, chaque syntagme dépendant toujours de l'élément situé à sa droite, le plus important est de savoir quel syntagme de la partie postérieure au syntagme clé est coordonné avec ce syntagme clé. Mais si on cherche un syntagme ayant un caractère plus proche de celui du syntagme clé, le traitement s'arrête lors de l'examen des éléments de la ligne du syntagme clé de la matrice. Mais pour examiner la correspondance de deux zones plus étendues, on cherche le chemin ayant le score le plus élevé.

Avant d'étudier la méthode de calcul de score d'un chemin, il faut définir le terme branche et d'autres notions liées à ce terme.

Définition 30 (Branche)
On appelle branche la partie de chemin entre deux éléments. L'élément d'une branche situé du côté du point de départ est appelé début, et l'autre élément est dit arrivée.

On tient compte uniquement des branches se dirigeant vers la gauche (horizontalement, ou vers le haut).

Le score d'un chemin quelconque est la somme des points de tous les éléments se trouvant sur ce chemin, calculée en tenant compte des conditions suivantes :

1. Si la branche se dirige vers la gauche horizontalement, on ne tient pas compte, dans le score, des points de l'arrivée de la branche. Par exemple, la branche $a \rightarrow b$ de la figure 5.8 (voir page suivante) calcule la correspondance entre E et H et celle entre E et I. Mais on définit comme principe que « on ne tient compte que de la correspondance entre un syntagme et un autre syntagme ». Pour la branche a, on ne tient donc compte que de la correspondance entre E et I, en considérant que H n'a pas de syntagme correspondant ;
2. Si la branche se dirige vers la gauche horizontalement, le score s est : $s = s - 2$;
3. Si la branche se dirige vers la gauche, vers le haut et que le début et l'arrivée de la branche sont éloignés de plus d'une ligne, le score s est : $s = s -$ ((différence de ligne $- 1$) $\times 2$). Ce qui signifie qu'est préférable une structure de coordination dans laquelle deux éléments coordonnés ont un nombre sensiblement identique de syntagmes.

Il est préférable de réduire les possibilités d'élargissement de la zone de la structure de coordination lorsque le syntagme contient un élément marquant une limite de structure de coordination. Pour cela, on attribue une pénalité au score du chemin si un syntagme contenant un élément de niveau plus élevé

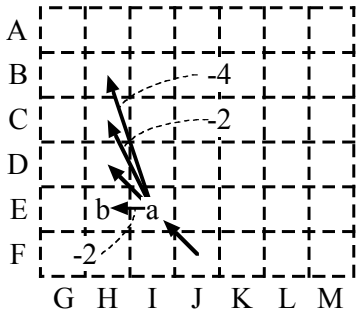


FIG. 5.8 – Calcul du score

que le niveau de limitation du syntagme clé est compris dans la structure de coordination. La pénalité p est calculée comme suit :

$$p = (\text{différence de niveau} + 1) \times 7$$

Cependant, cette pénalité est annulée si deux syntagmes de niveau élevé et de même type[25], différent du syntagme clé, ont une correspondance à l'intérieure de la structure de coordination.

Les pénalités sont prises en compte au cours du calcul de score du chemin de la façon suivante (voir la figure 5.9 (voir page suivante)) :

[1] **Si** la branche se dirige vers la gauche horizontalement, on calcule la pénalité pour le syntagme s_2 de l'arrivée de la branche (s_1, s_2) ;

[2] **Si** la branche se dirige vers la gauche et vers le haut,
alors

(a) **Si** l'annulation de pénalité n'est pas applicable, suite à l'examen de type des deux syntagmes s_1, s_2 de l'arrivée de la branche (s_1, s_2),
alors appliquer la pénalité à ces deux syntagmes ;

(b) **Si** le début et l'arrivé de la branche sont éloignés de plus d'une ligne,
alors appliquer la pénalité aux syntagmes situés à gauche de la branche et compris dans la structure de coordination sans être coordonnés avec un autre syntagme, c'est-à-dire aux syntagmes situés entre les syntagmes s_1 et s_2, où le début de la branche est en (s_1, s'_1) et l'arrivée est en (s_2, s'_2)

Il existe également des éléments utiles pour trouver la fin de l'étendue de la structure tels que など (etc), といった (tel que). De tels éléments sont appelés mot de bonus. Pour mieux exploiter cette caractéristique, on définit une règle comme suit :

[25] Des syntagmes sont de même type si tous leurs éléments – sauf le lexique de *jiritsugo* – sont identiques, c'est-à-dire si la catégorie et la forme de conjugaison de *jiritsugo*, ainsi que tous les *fuzokugo* sont identiques.

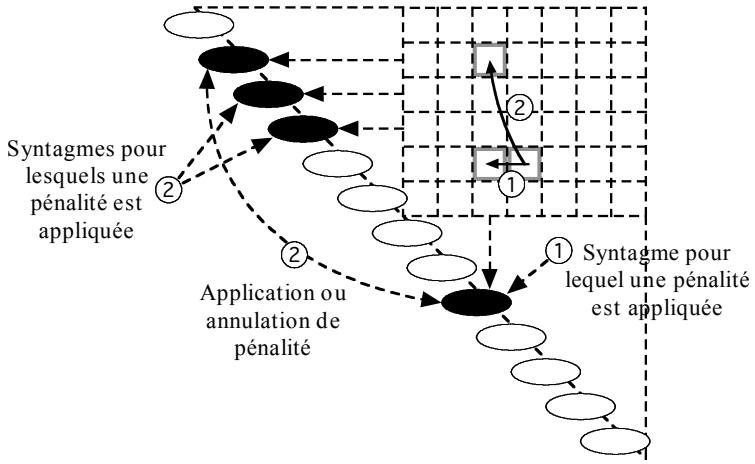


FIG. 5.9 – Calcul des pénalités

[1] Soit (s_1, s_2) le point de départ d'un chemin c,
Si le syntagme s_2 ou le syntagme suivant contient un élément défini comme mot de bonus,
alors le score s du chemin c est : $s = s + 6$.

On calcule ce score avec une méthode de programmation dynamique, c'est-à-dire en réalisant le calcul des colonnes une par une de gauche à droite, on calcule pour chaque colonne le meilleur chemin de chaque élément. Ce processus, représenté sur la figure 5.10 (voir page suivante), se déroule comme suit :

Soient m la matrice partielle, l le nombre de lignes de m et c le nombre de colonnes de m.
Pour calculer le meilleur chemin allant du point de départ (s_0, s_n) à l'élément (s_i, s_j), on applique l'algorithme suivant :

[1] $k := 0$

[2] Répéter indéfiniment

(a) $ch[k]$:= score du meilleur chemin de (s_0, s_n) à (s_{0+k}, s_{j-1}) + score de la branche de (s_{0+k}, s_{j-1}) à (s_i, s_j) ;

(b) Si k = i, alors arrêter de répéter ;

(c) $k := k + 1$

Si $ch[t]$ est l'élément ayant le score plus élevé, alors le meilleur chemin de (s_0, s_n) à (s_t, s_{j-1}) prolongé à (s_i, s_j) est le meilleur chemin de (s_0, s_n) à (s_i, s_j).

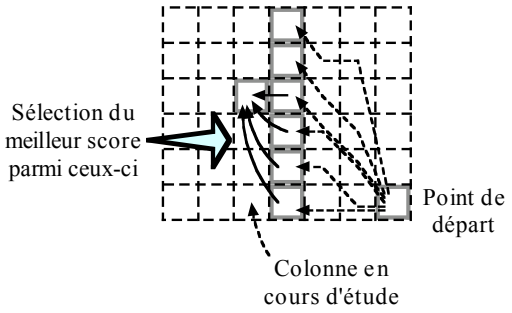


FIG. 5.10 – Méthode de programmation dynamique

Lorsqu'on a calculé tous les meilleurs chemins partant d'un point de départ donné, et arrivant à chaque élément de la colonne la plus à gauche de la matrice partielle, on considère le chemin ayant le score le plus élevé comme le meilleur chemin partant de ce point de départ.

Parmi les meilleurs chemins de chaque point de départ, on considère le chemin ayant le score le plus élevé comme le chemin indiquant la structure de coordination.

5.4.5 Analyse des relations de dépendance

Cette étape consiste en la reconnaissance des relations de dépendance dans l'ensemble de la phrase sans trahir la structure de coordination détectée. En d'autres termes, on cherche pour chaque syntagme de la phrase, le syntagme dont il dépend. Ce traitement est basé sur 38 règles réparties en 5 catégories selon le type de dépendance du syntagme pour lequel on cherche son récepteur.

- catégorie des exceptions : regroupe 5 règles d'exception ;
- catégorie de l'apposition : regroupe 3 règles pour les syntagmes ayant la propriété appositive ;
- catégorie des arguments : regroupe 12 règles pour les syntagmes ayant la propriété d'arguments ;
- catégorie des *rentai* : regroupe 5 règles pour les syntagmes ayant la propriété *rentai* ;
- catégorie des *renyô* : regroupe 13 règles pour les syntagmes ayant la propriété *renyô*.

Dans ces fichiers, les règles sont définies sous la forme de :

$$(\; SD \; (\; [SR_1 \; TR_1] \; [SR_2 \; TR_2] \; \ldots \; [SR_n \; TR_n] \;) \; Lim \; P \;)$$

où

- SD, types de syntagme dépendant à traiter ;
- SR_i, types de syntagme pouvant recevoir le syntagme de type SD ;

– TR_i, types de réception tels que coordination (P), apposition (A) et autres (D) ;
– Lim, types de limite de zone que la relation de dépendance ne peut pas franchir ;
– P, position prioritaire de récepteur et $P \in \mathbb{Z}$.

Afin de trouver pour un syntagme *sd* donné d'une phrase le syntagme *sr* dont il dépend, on choisit la catégorie de règles concernée par le syntagme *sd*. On cherche ensuite des syntagmes correspondant aux types définis dans les règles en examinant chaque syntagme à partir de celui situé juste après le syntagme *sd*. Lorsqu'on rencontre un syntagme correspondant aux types de limite de zone définis dans les règles, et si on a déjà trouvé des syntagmes candidats à une position plus proche du syntagme *sd*, on ne tient plus compte d'aucun syntagme situé après ce syntagme, y compris de ce syntagme lui-même. Si les types de limite de zone ne sont pas définis, tous les syntagmes sont considérés comme limite, c'est-à-dire qu'on arrête la recherche dès la découverte du premier syntagme candidat.

En ce qui concerne la position prioritaire du récepteur, si elle vaut 1, le syntagme récepteur *sr* est le syntagme candidat le plus proche du syntagme *sd*, si elle vaut 2, le syntagme *sr* est le deuxième syntagme candidat en comptant à partir de celui le plus proche. La position -1 signifie que le syntagme récepteur *sr* est celui qui se trouve le plus loin du syntagme *sd*.
Par exemple, la règle

(((係 ; デ格　読点) (係 ; カラ格　読点) (係 ; マデ格　読点))
([((用言 ; 強)) D])
((レベル ; C) (レベル ; B'))
2)

(((dép : cas-*de* virgule)(dép : cas-*kara* virgule)(dép : cas-*made* virgule))
([((*yôgen* : fort)) D])
((niveau : C) (niveau : B'))
2)

signifie que le syntagme ayant comme propriété de dépendance « cas de *de* comportant une virgule » (係 ; デ格　読点), « cas de *kara* comportant une virgule » (係 ; カラ格　読点) ou « cas de *made* comportant une virgule » (係 ; マデ格　読点), dépend du syntagme ayant comme propriété « *yôgen* de type fort » (用言 ; 強) avec une relation de type D (c'est-à-dire une relation de dépendance normale). Si on rencontre un syntagme ayant comme propriété レベル ; C (niveau de limitation : C) ou レベル ; B' (niveau de limitation : B'), on ne tient pas compte des syntagmes situés après lui, et le syntagme candidat prioritaire est celui qui se trouve en deuxième position à partir du syntagme dépendant. Ainsi, si le syntagme en cours de traitement a la propriété « cas de *de* comportant une virgule » (係 ; デ格　読点), on applique cette règle et on recherche d'abord des syntagmes ayant comme

propriété « *yôgen* de type fort » (用言 ; 強) et situés après lui. Supposons que nous ayons rencontré deux syntagmes candidats et trouvé un syntagme ayant comme propriété « niveau de limitation : B' » (レベル ; B'), on arrête alors l'analyse, établissant ainsi la relation de dépendance entre le syntagme à traiter et le syntagme candidat en deuxième position qui gouverne donc ce premier.

5.4.6 Résultat d'analyse

Après avoir obtenu toutes les possibilités de relations de dépendance de la phrase, le système les examine de manière à choisir le résultat le plus probable. L'évaluation du résultat est réalisée essentiellement en fonction de la distance entre chaque élément des relations de dépendance et en fonction du degré de satisfaction des arguments au niveau de la surface.

Le système propose également de réaliser une analyse de la structure argumentale, que nous ne traitons pas dans le présent mémoire. Cette analyse consiste à chercher pour chaque résultat un modèle correspondant dans le dictionnaire de structures argumentales, pour déterminer le cas profond de chaque élément. Le système choisit ensuite comme résultat le plus probable le résultat dont la correspondance avec le modèle est la plus élevée.

5.5 Analyse des résultats

5.5.1 Introduction

Cette dernière section est consacrée à l'étude des résultats d'analyse obtenus par chaque système et à l'analyse de leurs points forts et de leurs faiblesses.

5.5.2 Installation des analyseurs

Environnement d'installation

Ordinateur :	Apple iBook
Processeur :	G3 466MHz
Mémoire :	192Mo
Système d'exploitation :	MacOS X, ver. 10.2.6
Outils de développement :	Developer Tools, ver. December 2002

Outils et dictionnaires nécessaires

1. Utilitaires informatiques
2. Analyseur morphologique
3. Dictionnaires et autres données
4. Grammaire et autres règles grammaticales

	SAX Vers. 2.1	MSLR Vers. 1.04	KNP Vers. 2.0 b6
1	SICStus Prolog Vers.3		nkf, jperl
2	JUMAN		JUMAN
3		Dict. du système (basé sur l'EDR)	Dict. du japonais EDR Table des lexiques[26] Dict. IPAL (*Case frame*)
4	Gram. du système (à titre d'exemple)	Gram. du système Table de connexions	

Quelques remarques sur l'installation de chaque analyseur seront présentées dans cette section. L'installation a été considérée comme un succès si tout d'abord toutes les étapes de compilation se sont déroulées correctement et si l'analyseur est capable d'analyser la phrase élémentaire :

太郎は東京駅で電車に乗る « Tarô prend le train à la gare de Tôkyô »

Installation de SAX

Nous avons utilisé la version d'essai de SICStus Prolog Version 3.10.1 pour Power PC Darwin 6.4 (licence d'essai valable 1 mois).

L'installation a été réalisée de manière à ce que le système fonctionne dans une configuration avec analyse morphologique par JUMAN, en utilisant le fichier de configuration prédéfini appelé `sax_user.JUMAN.pl`.

La version la plus récente de JUMAN (Version 3.61) a été installée, mais celle-ci ne contenant pas le module d'interfaçage Prolog (appelé `juman_pl`), il a été récupéré d'une ancienne version (la 3.5).

Quelques modifications ont été nécessaires, comme décrit ci-après :

1. JUMAN
   - Dans le fichier Prolog `socket.pl` : changement du nom du prédicat `socket_close/1` car ce nom est celui d'un prédicat prédéfini.
   - Dans le fichier C `juman_p.c` :
     - changement, dans la définition des macros, de la valeur de `SERV_TCP_PORT` de 16010 en 32000 (valeur par défaut dans la version 3.6) ;
     - dans la définition des variables globales, ajout d'un « ; » à la fin de la ligne `extern FILE *Jumanrc_Fileptr` ;
     - dans la fonction `void juman_out_pl(int newsockfd)`, modifier l'appel à la procédure `_print_all_mrph (p_buffer_num-1)` en `_print_all_mrph(NULL, p_buffer_num-1)` (le fichier `juman_p.c` étant dans une version plus ancienne, les modifications apportées aux fichiers de la version 3.6 n'ont pas été reportées dans celui-ci).
   - Dans le fichier `Makefile` : ajout du drapeau `-no-cpp-precomp` dans la ligne de définition `DEFS`. Ce drapeau doit également être ajouté dans le `Makefile` du dossier `dic`, sur la ligne `gcc -E -P [...]` .

[26] Il s'agit de la Table des lexiques catégorisés produite par le 国立国語研究所, *The national language research institute.*

2. SAX
   - Dans le fichier `.sicstusrc` créé dans le répértoire racine, définition du traitement des prédicats non définis de manière à provoquer `fail`, avec `:- unkown(_, fail).` (La nécessité de cette spécification est signalée non pas dans le manuel mais sur le site internet de SAX).
   - Dans les fichiers de démarrage (`Run_JUMAN.pl` par exemple) consultant les fichiers de configurations personnalisées et le module `sax_trans` pour traduire la grammaire en clauses SAX :
     - ajout de « `.pl` » à la fin du nom du fichier consulté comme suit : `consult(library('sax_user/sax_user.JUMAN.pl'))` ;
     - écriture du nom de chemin absolu du fichier contenant la grammaire tel que : `sax_consult('/Users/yayoi/Documents/SAX201/grammar/japanese')`.
       Si on utilise `library_directory` comme spécifié dans le manuel, SICStus transforme le nom de chemin avec son répertoire `library` au moment de l'appel du prédicat `absolute_file_name`.

Avec les modifications ci-dessus, le système n'est pas encore capable de réaliser l'analyse syntaxique de notre phrase d'exemple. En effet, à partir du résultat de l'analyse morphologique, le système conserve comme donnée la catégorie lexicale de chaque mot. Or, dans la grammaire, tous les *joshi* (particules) sont traités avec leur sous-catégorie lexicale, c'est-à-dire *kaku-joshi* (particule de cas) ou *shû-joshi* (particule finale), etc. Il faut donc modifier, soit la grammaire soit la procédure de traitement des données morphologiques. Afin de rendre le système capable de réaliser une analyse syntaxique du japonais au moins pour des phrases simples et ce sans trop modifier les codes initiaux, j'ai modifié la procédure `getMorphInf/4`, qui reçoit l'ensemble des données d'un morphème donné et qui renvoit sa catégorie lexicale, sa forme canonique et d'autres informations, comme suit :

```
% getMorphInf(+Morph, -Hinsi, -Midasi, -Args)
%
% À partir de l'ensemble des données Morph,
% on extrait la catégorie Hinsi, la forme canonique Midasi,
% et d'autres informations Args.
%

% Modif Y. DELLOYE (15/06/03) :
% Si Hinsi est 助詞(particule), on retourne 分類名(sous-cat).
%
getMorphInf(Morph, Hinsi, Midasi, Args) :-
     juman :get_hinsi(Morph, Joshi),
     Joshi == 助詞,!,
     juman :get_bunrui(Morph, Hinsi),
     juman :get_midasi(Morph, Midasi),
```

```
    Args = [].


getMorphInf(Morph, Hinsi, Midasi, Args) :-
    juman:get_hinsi(Morph, Hinsi),
    juman:get_midasi(Morph, Midasi),
    Args = [].
```

Avec ces modifications, le système réalise enfin l'analyse de la phrase d'exemple décrite précédemment.

Installation de MSLR

L'installation de MSLR est relativement simple, puisqu'il suffit de modifier les paramètres liés au fait que le système d'exploitation de notre ordinateur est basé sur un Unix de type BSD, comme suit :

- modification du fichier `Makefile` afin d'utiliser les drapeaux correspondant à LINUXPPC ;
- dans le répertoire `sufary`, fichier `s1.c`, changement de `<malloc.h>` en `<sys/malloc.h>` ;
- dans le répertoire `parser`, fichier `mslr.c`, changement de `<getopt.h>` en `<kpathsea/getopt.h>` ;
- création du fichier `.mslrrc` pour positionner les variables d'environnement MSLR_GRM, MSLR_TBL et MSLR_DIC ;
- enfin, LaTeX étant installé sur notre ordinateur de test, il est nécessaire de donner un nouveau nom à l'exécutable de création de l'index de dictionnaire (`mkindex` par défaut), car celui-ci est déjà utilisé par la fonction « make index » de LaTeX. Cette modification se fait dans le fichier `Makefile` du répertoire `sufary`.

Installation de KNP

Du fait de la difficulté d'acquisition de l'ensemble des outils nécessaires (le dictionnaire EDR japonais coûte environ 500 euros pour une utilisation académique par un laboratoire, et 15 000 euros pour un particulier, et pour la table de lexique il faut contacter directement l'éditeur ou le laboratoire de 国立国語研究所, *The national language research institute*), nous utiliserons le système disponible sur le site internet :
`http://www.kc.t.u-tokyo.ac.jp/nl-resource/knp.html`.

5.5.3 Corpus d'analyse

Nous avons choisi cinquante phrases pour analyser les résultats, considérant cette quantité comme un nombre raisonnable pour notre étude.

Critères de choix du corpus

Afin de mieux nous rendre compte des points forts et faibles de chaque analyseur, nous avons choisi cinq ouvrages de nature différente tels que :

1. un ouvrage spécialisé[27] dans un domaine scientifique pur ;
2. un ouvrage spécialisé dans un domaine des sciences humaines ;
3. un brevet technique ;
4. une œuvre littéraire ;
5. un article de quotidien.

Les ouvrages ci-dessous ont été choisis selon ces critères :

1. 『形式言語とオートマトン入門』小倉久和、1996 (*Initiation aux langages formels et aux automates*, Hisakazu OGURA, 1996) : manuel avec des exercises visant les étudiants de premier cycle universitaire ;
2. 『文法 II』大野晋、柴田武、1977 (*Grammaire II*, Susumu Ôno, Takeshi SHIBATA, 1977) : septième tome de la collection 「岩波講座日本語」(Cours d'Iwanami : japonais). Cette collection de référence couvre à peu près la totalité des sujets de linguistique japonaise. Le septième tome est dédié aux catégories de *fuzokugo*, à savoir *joshi* et *jodôshi* ;
3. 「電気掃除機の集塵袋」(平3-186241)、松下電器産業株式会社 (出願者)、1991 (公開) (*Sac à poussière pour aspirateur électrique* (Hei 1-327999), Matsushita Electric industrial Co.,Ltd. (Demandeur), 1991 (Publication)) : texte de brevet, contenant beaucoup de termes chimiques ;
4. 『国境の南、太陽の西』村上春樹、1995 (*Sud de la frontière, ouest du soleil*, Haruki MURAKAMI, 1995) : MURAKAMI est un des écrivains contemporains les plus populaires à l'heure actuelle ;
5. 「朝日新聞」 (Journal ASAHI) : un des trois grands quotidiens japonais.

Nous allons maintenant étudier de plus près chacun de ces ouvrages. Nous commençons par « *Initiation aux langages formels et aux automates* » et terminerons par l'article du journal ASAHI. Les phrases extraites sont en général les dix premières du livre afin de ne pas sélectionner sciemment celles qui ont une structure simple ou complexe. Toutefois, lorsque les dix premières

[27] On considère comme ouvrage spécialisé un livre dont le public est limité au moins aux étudiants de l'enseignement supérieur.

phrases ont une construction très particulière, telle que des énumérations, les dix premières phrases d'un chapitre quelconque sont retenues. Concernant l'article du Journal ASAHI, la première page du site officiel[28] a été choisie. Chaque section commence par la présentation des phrases extraites avec leur traduction, puis les premiers constats sur les caractéristiques de leur structure syntaxique.

Initiation aux langages formels et aux automates

3.1 離散グラフ P49 (3.1 Graphe discret)

1. 離散グラフ（discrete graph）は，節点（ノード，node）の集合と節点を結ぶ辺（エッジ，edge）の集合で，単にグラフともいう。
(Les graphes discrets sont constitués d'un ensemble de nœuds et d'un ensemble d'arêtes reliant ces nœuds. Ils sont également appelés simplement graphes.)
2. 有限個の節点と有限個の辺からなる離散グラフが有限グラフ（finite graph）である。
(Un graphe constitué d'un nombre fini de nœuds et d'un nombre fini d'arête est appelé graphe fini.)
3. 関数を表現するのにグラフがよく用いられるが，それは節点が連続無限集合のグラフとみなせる。
(Les graphes sont souvent utilisés pour représenter des fonctions. Ces graphes peuvent en fait être considérés comme des graphes d'un ensemble infini continu de nœuds.)
4. 節点の集合をＶとする。
(Soit V l'ensemble des nœuds.)
5. 節点は，頂点（vertex），点（point）などともいう。
(Un nœud est également appelé sommet ou point.)
6. 隣接する節点をつなぐ辺の集合をＥとする。
(Soit E l'ensemble des arêtes reliant des nœuds voisins.)
7. ＥはＶ×Ｖの部分集合，Ｅ ∈ Ｖ×Ｖ，である。
(E est un sous-ensemble de $V \times V$, et $E \in V \times V$.)
8. 辺は，弧（アーク，arc）などともいう。
(Une arête est également appelée arc.)
9. 辺でつながれた節点をたがいに隣接点という。
(Des nœuds reliées par une arête sont appelés nœuds voisins.)
10. グラフはＧ＝（Ｖ，Ｅ）で示される。
(Un graphe est représenté par $G = (V, E)$.)

[28] `http://www2.asahi.com`

Premiers constats En plus de la présence de formules mathématiques, ce texte est caractérisé par un nombre important de parenthèses servant à introduire les termes anglais correspondants. La structure elle-même étant assez simple, le problème pour un analyseur automatique se trouve essentiellement dans le traitement de ces deux types de séquences spécifiques. On peut poser comme hypothèse que l'analyseur peut résoudre le problème des formules mathématiques en les considérant comme un bloc de mot inconnu de type substantif.

N° de phrase	1	2	3	4	5	6	7	8	9	10	Moy.
Nb de caractères	48	33	44	11	18	20	21	20	21	17	25,3
Nb de propositions	2	2	3	1	1	3	1	1	2	1	1,7

Tab. 5.2 – Initiation aux langages formels et aux automates

Grammaire II

2. 助動詞１ (Section 2. *jodôshi* 1)

1. 助動詞とよばれている語群は、述語の部分の構成に関与するもので、述語の中心となる自立語ー主として動詞ーを補助するはたらきをもち、その補助機能がある程度定式化されているものの集合体である。
   (L'ensemble des mots appelés *jodôshi* est l'ensemble des unités intervenant dans la construction d'une partie du prédicat, qui ont pour fonction d'assister le *jiritsugo*, élément principal du prédicat – en particulier les verbes –, et pour lesquelles ces fonctions auxiliaires sont à un certain degré formalisées.)
2. これらの語の補助機能の質を検討して、助動詞とよぶべき語の範囲を限定することも試みられている。
   (Certains essaient de définir l'étendue de l'ensemble des mots que l'on peut appeler *jodôshi*, en examinant la nature de ces fonctions auxiliaires des mots.)
3. ここではそうした問題も考慮に入れながら、一応これまで普通に助動詞とよばれているものを考察の対象にしていくことにする。
   (Dans cette section, en tenant compte de ce genre de problèmes, nous considérons, pour le moment, comme objet de réflexion les mots appelés ordinairement *jodôshi* jusqu'à maintenant.)
4. このような意味での助動詞に属する語は、平安時代では二〇語あまりにのぼる。
   (Les mots appartenant dans ce sens à la catégorie *jodôshi* s'élevaient à une vingtaine à l'époque de Heian.)
5. これらは普通「活用・接続・意味」の三方面から説明されることが多かった。

(Il était généralement courant d'expliquer ces mots selon trois aspects : variations morphologiques, connexion et sens.)

6. ことに、いわゆる古典文法では、古典を解釈するという目的に支えられてきた歴史的事情もあって、意味の研究に主力が注がれてきた。
(En particulier, pour la grammaire de l'ancien japonais, les principaux efforts étaient d'autant plus concentrés sur les recherches sémantiques que cette grammaire était historiquement basée sur un objectif de compréhension des ouvrages classiques.)
7. しかし、文法研究は文の構成に関する法則を求めるものであるから、助動詞についても、それが述語の部分を構成するにあたってどんな役割を果たしているか、さらには文全体の構成にどう関わるものであるかを問題としなければならないであろう。
(Cependant, les recherches grammaticales cherchant à découvrir les règles concernant la structure des phrases, il faudrait, pour les *jodôshi* également, poser des questions telles que : quel rôle jouent-ils dans la construction d'une partie du prédicat, ou encore dans quelle mesure participent-ils à la formation d'une phrase ?)
8. それには、文を構成する場合、助動詞がたがいに承接する順序に一定の法則があることを、助動詞の構文上のはたらきを考える上で重要な事実として、まず認めるべきである。
(Pour cela, il est indispensable de reconnaître tout d'abord comme une réalité très importante pour réfléchir sur les fonctions syntaxiques des *jodôshi*, qu'il existe certaines règles sur l'ordre avec lequel les *jodôshi* se succèdent lors de la construction d'une phrase.)
9. そして、これを軸として、意味・活用・接続の研究をこれに関連させ、総合的に把握していくことが、助動詞研究のあるべき方向だと私は考える。
(Sur la base de cette réalité et en y associant des recherches sur leur sens, leurs variations morphologiques et leur connexion, on essaie de saisir globalement les *jodôshi*, ce qui est, me semble-t-il, une orientation correcte pour les recherches sur les *jodôshi*.)
10. 以下、このような立場から、奈良時代・平安時代の助動詞を相互承接と意味との関連を中心に考察していきたい。
(Nous réfléchissons ci-après ; en adoptant une telle position, aux *jodôshi* de l'époque Nara et de l'époque Heian, en traitant essentiellement les rapports entre la connexion mutuelle de ces unités et leurs sens.)

Premiers constats Les phrases sont relativement longues, donc complexes, mais elles ne sont constituées, contrairement aux phrases de l'ouvrage précédent, que d'unités « normales » sans symboles, ni *rômaji*. Il n'y a même presque aucun chiffre. On constate de nombreuses occurrences des

mots démonstratifs, mais cela ne pose pas de problèmes dans une analyse syntaxique.

N° de phrase	1	2	3	4	5	6	7	8	9	10	Moy.
Nb de caractères	92	46	58	36	35	61	112	79	66	51	63,6
Nb de propositions	6	4	2	2	2	4	8	5	5	1	3,9

TAB. 5.3 – Grammaire II d'Iwanami

Brevet

1. 口芯と紙袋からなり電気掃除機の集塵室に着脱自在にセットされる集塵袋であって、植物精油からなる芳香剤を精油非透過性物質からなるフィルム状の材料で包んだ状態で集塵袋の一部に取り付け、集塵袋の電気掃除機へのセット時に、あるいは電気掃除機使用時に前記フィルム状の材料を破るように構成した電気掃除機の集塵袋。
   (Sac à poussière pour aspirateur électrique, composé d'une collerette et d'un sac en papier, et qui se fixe librement dans le compartiment à poussière de l'aspirateur électrique, et sur une partie duquel est fixé un aromatisant composé d'huile végétale raffinée et enrobé de matériaux en film composés de matière imperméable à l'huile raffinée, et conçu pour que les matériaux en film décrits précédemment soient déchirés lors de la fixation du sac à poussière sur l'aspirateur électrique ou lors de l'utilisation de l'aspirateur électrique.)
2. 本発明は電気掃除機の集塵袋に関するものである。
   (L'invention concerne un sac à poussière pour aspirateur électrique.)
3. 従来のこの種電気掃除機の集塵袋としては、ダニ類を殺す殺虫剤やカビの繁殖を防ぐ殺菌剤を紙袋に含浸させたものが一般的に知られている。
   (Parmi les sacs à poussière antérieurs pour aspirateur électrique, sont généralement connus ceux dont le sac en papier est imprégné d'insecticide tuant les acariens et de bactéricide empêchant le développement de moisissures.)
4. このように紙袋に殺虫剤や殺菌剤を含浸させてなる集塵袋を用いて殺虫、殺菌を行う以外に、電気掃除機の空気流通路より集塵袋内に植物精油を含浸させた多孔質基材を吸い込ませて収納させ、前記植物精油からの芳香により集塵袋内のダニ類やカビの殺虫、殺菌を行わせるようにしたものもある。
   (En plus de tuer les insectes et d'éliminer les bactéries à l'aide d'un sac à poussière constitué d'un sac en papier imprégné d'insecticide et de bactéricide comme décrit précédemment, il en existe également qui, par stockage via aspiration par le flux d'air de l'aspirateur électrique dans le sac à poussière d'un matériau de base poreux imprégné d'huile

végétale raffinée, sont conçus pour permettre de tuer et d'éliminer les acariens et les moisissures présents à l'intérieur du sac à poussière grâce à l'odeur émise par l'huile végétale raffinée décrite précédemment.)

5. ところで上記集塵袋に含浸させる殺虫剤や殺菌剤は農薬が一般に使用され、電気掃除機の運転時に大気中に殺虫剤や殺菌剤が微量ながら放出されるため、人体に悪影響を及ぼす危険性があった。
(Cependant, des produits chimiques agricoles étant généralement utilisés pour l'insecticide et le bactéricide imprégnés dans les sacs à poussière décrits précédemment, il existait des risques de mauvaise influence sur l'homme du fait que l'insecticide et le bactéricide sont, ne serait-ce que peu, émis dans l'air lors de l'utilisation de l'aspirateur électrique.)
6. また、上記のように植物精油を含浸させたものであると、使用時に揮発し、使用時の効果が薄れるという問題があった。
(Par ailleurs, pour ceux imprégnés d'huile végétale raffinée comme décrit précédemment, il y avait des problèmes de perte d'efficacité due à une volatilisation avant utilisation.)
7. さらに、植物精油を含浸させた多孔質基材を電気掃除機で集塵袋に吸い込ませるものでは、その吸い込ませる作業が面倒であるという問題があった。
(En outre, pour les sacs à poussière dans lesquels on aspire un matériau de base poreux imprégné d'huile végétale raffinée avec un aspirateur électrique se posait le problème de la pénibilité de cette opération d'aspiration.)
8. 本発明はこのような課題を解決するもので、人体への影響をなくし、また使用前の揮発による殺虫、殺菌効果の低下をなくし、さらに多孔質基材を吸い込ませるような作業を不要とし、作業性を向上できるようにすることを目的とするものである。
(L'invention résout ces problèmes et a pour objectif de permettre la suppression de l'influence sur l'homme d'une part, de la perte d'effets insecticide et stérilisant due à une volatilisation avant utilisation d'autre part, et d'améliorer la qualité d'opération sans nécessiter des opérations telles que l'aspiration d'un matériau de base poreux.)
9. この課題を解決するために本発明は、口芯と紙袋からなり電気掃除機の集塵室に着脱自在にセットされる集塵袋であって、植物精油からなる芳香剤を精油非透過性物質からなるフィルム状の材料で包んだ状態で集塵袋の一部に取り付け、集塵袋の電気掃除機へのセット時に、あるいは電気掃除機使用時に前記フィルム状の材料を破るように構成したものである。
(Afin de résoudre ces problèmes, l'invention est un sac à poussière composé d'une collerette et d'un sac en papier, et qui se fixe librement dans le compartiment à poussière de l'aspirateur électrique, et sur une partie duquel est fixé un aromatisant composé d'huile végétale raffinée et enrobé de matériaux en film composés de matière imperméable à l'huile

raffinée, et conçu pour que les matériaux en film décrits précédemment soient déchirés lors de la fixation du sac à poussière sur l'aspirateur électrique ou lors de l'utilisation de l'aspirateur électrique.)

10. この構成により、集塵袋を電気掃除機にセットしたり電気掃除機を使用するときに前記フィルム状の材料を破り、植物精油の芳香を発し、集塵袋に集められたごみの中のダニを殺すことができ、同時にごみの中の細菌の繁殖を抑えることができ、排気と一緒に悪臭が出ることもない。
(Avec cette structure, les matériaux en film décrits précédemment étant déchirés lors de la fixation du sac à poussière sur l'aspirateur électrique ou lors de l'utilisation de l'aspirateur électrique, et l'odeur de l'huile végétale raffinée se propageant, on peut tuer les acariens qui se trouvent dans les poussières ramassées à l'intérieur du sac à poussière, et empêcher en même temps le développement des bactéries dans la poussière, sans que ne se dégagent de mauvaises odeurs avec l'échappement.)

Premiers constats Les phrases sont extrêmement longues et complexes. La plupart de ces phrases contiennent des structures de coordination, pour lesquelles, même pour un être humain, il est difficile de déterminer les syntagmes coordonnés.

Toutefois, ce texte représentant le style typique des brevets, il est impératif, si on conçoit par exemple un analyseur syntaxique des brevets techniques, de créer un système capable d'analyser ce genre de structures sans problèmes.

N° de phrase	1	2	3	4	5	6	7	8	9	10	Moy.
Nb de caractères	149	23	64	134	87	54	67	111	162	127	97,8
Nb de propositions	9	2	4	8	5	5	5	11	11	10	7,0

TAB. 5.4 – Brevet

Roman de MURAKAMI

1. 僕が生まれたのは一九五一年の一月四日だ。
(Je suis né le 4 janvier 1951.)
2. 二十世紀の後半の最初の年の最初の月の最初の週ということになる。
(C'est donc la première semaine du premier mois de la première année de la deuxième moitié du vingtième siècle.)
3. 記念的といえば記念的と言えなくもない。
(On peut dire que c'est un jour mémorable.)
4. そのおかげで、僕は「始（はじめ）」という名前を与えられることになった。

(C'est la raison pour laquelle on m'a donné le prénom « Hajime (début) ».)

5. でもそれを別にすれば、僕の出生に関して特筆すべきことはほとんど何もない。
(Mais, à part ce fait, il n'y a rien de particulier concernant ma naissance.)
6. 父親は大手の証券会社に勤める会社員であり、母親は普通の主婦だった。
(Mon père était salarié d'une grande maison de titres, et ma mère était une femme au foyer ordinaire.)
7. 父親は学徒出陣でシンガポールに送られ、終戦のあとしばらくそこの収容所に入れられていた。
(Envoyé à Singapour du fait de la mobilisation des étudiants (*N.d.T. : lors de la seconde guerre mondiale*), mon père y resta dans un camp pendant un certain temps après la fin de guerre.)
8. 母親の家は戦争の最後の年にＢ２９の爆撃を受けて全焼していた。
(La maison de ma mère fut entièrement brulée par un bombardement de B29 la dernière année de la guerre.)
9. 彼らは長い戦争によって傷つけられた世代だった。
(Ils étaient la génération des gens blessés par cette longue guerre.)
10. でも僕が生まれた頃には、もう戦争の余韻というようなものはほとんど残っていなかった。
(Mais, quand je suis né, il n'y avait presque plus de résonance de la guerre.)

Premiers constats La structure de phrases est relativement simple, ce qui est probablement dû au style de l'auteur. On ne peut pas généraliser ce constat à tous les romans japonais.

Une phrase ne comprend que deux ou au maximum trois propositions, mais un syntagme nominal est parfois assez long (concaténation répétitive des substantifs par la particule *no*, équivalent *grosso modo* à la préposition française « de »). Les structures sont particulièrement stables avec relativement peu d'omission d'éléments : la plupart des phrases sont bien constituées d'un syntagme thématisé par la particule *wa* et d'un syntagme rhème comprenant un prédicat.

N° de phrase	1	2	3	4	5	6	7	8	9	10	Moy.
Nb de caractères	20	31	19	35	36	33	43	30	23	41	31,1
Nb de propositions	2	2	1	2	3	3	2	2	2	2	2,1

Tab. 5.5 – Roman de Murakami

Article du Journal ASAHI

1. ＡＰ通信は１０日、イラク戦争で少なくとも３２４０人の民間人が死亡したとする独自調査の結果を報じた。
   (L'*Associated Press* a annoncé le 10 (juin) le résultat de sa propre enquête : on compte au moins trois mille deux cent quarante morts civils pendant la guerre contre l'Iraq.)
2. 同通信の記者が、全国１２４の病院のうち、ほとんどの主要病院を含む６０病院を訪れ、記録を調べたり、関係者に聞き取りを行ったりして死者数を集計した。
   (Ce chiffre a été calculé par les journalistes d'AP qui ont réalisé des recherches sur des enregistrements et interrogé les intéressés en se rendant dans soixante hôpitaux, incluant la presque totalité des principaux établissements, parmi les 124 répartis dans tout le pays.)
3. これによると、戦争が始まった３月２０日から、主な戦闘が終結した後の４月２０日までの間に、バグダッドでは１８９６人の民間人が死亡したとしている。
   (Ce rapport dit qu'il y a eu 1896 morts civils à Bagdad entre le 20 mars, début de la guerre, et le 20 avril, après les principales batailles.)
4. 集計では、軍人と民間人の死者を区別していない病院の記録は除外。
   (Les enregistrements des hôpitaux où il n'y avait pas de distinction entre les morts militaires et les morts civils, n'ont pas été pris en compte.)
5. 南部にあるイラク第２の都市バスラでは、三つの病院で計４１３人の死亡記録があり、医師たちは８５％が民間人とみているが、確認ができないため集計に含めなかった。
   (À Bassora, deuxième ville iraqienne située au sud, trois hopitaux ont enregistré 413 morts, dont 85% étaient, d'après les médecins, des civils, mais ils n'ont pas été comptés faute de moyen de vérification.)
6. このほか、病院に運ばれずに埋葬された例も多いとみられ、実際の民間人死者数はさらに数千人多い可能性もあるという。
   (Par ailleurs, il semble y avoir beaucoup de cas où les morts ont été enterrés sans avoir été transférés dans un hôpital. Le rapport signale qu'il est possible que les morts civils réels soient encore de quelques milliers de personnes en plus.)
7. 朝日新聞が４月に行った独自調査では、同１０日までにバグダッドでの民間人死者数が約１０００人という結果だった。
   (L'enquête réalisée par ASAHI au mois d'avril a présenté comme résultat que le nombre de morts civils à Bagdad jusqu'au 10 avril était d'environ mille personnes.)
8. メディア情報から民間人死者数を集計している国際プロジェクト「イラク・ボディー・カウント」は、戦闘終結後の不発弾による死

者などを含めて１１日現在、最低５５３１人、最高７２０３人としているが、病院で聞き取りをするＡＰ通信の集計方法はより確実性が高いとみられる。
(Le projet international « Irak Body Count » qui compte le nombre de morts civils à partir des informations médiatiques, considère le nombre de morts au 11 juin, comprenant les morts après la guerre comme ceux tués par les bombes non explosées, comme 5531 au minimum et 7203 au maximum. Cependant, la méthode de calcul de l'AP qui réalise des interrogations dans les hôpitaux est plus fiable.)

9. 一方、米英軍は民間人死者の推計をしていない。
(Par ailleurs, les armées américaine et britannique n'ont pas réalisé d'estimation des morts civils.)
10. ＡＰ通信によると、９１年の湾岸戦争での民間人死者数は、イラク当局の統計で２２７８人。
(Selon l'AP, le nombre de morts lors de la guerre du Golfe en 1991 était de 2278 d'après les statistiques nationales iraqiennes.)

Premiers constats La structure des phrases est très brève avec beaucoup d'omission de particules et de copules et avec un nombre important d'occurrences de chiffres. Ce qui est sans doute dû à l'objectif bien défini de transmission des informations et ce dans un espace très limité. Une analyse de ce style de texte nécessite la définition d'une grammaire tenant compte de l'ensemble de ces types de phénomènes d'omissions des mots outils.

N° de phrase	1	2	3	4	5	6	7	8	9	10	Moy.
Nb de caractères	49	72	71	31	77	55	54	128	22	42	60,1
Nb de propositions	3	4	4	2	5	5	3	6	1	1	3,4

Tab. 5.6 – Article du quotidien Asahi

Nous passons maintenant à l'analyse des résultats obtenus par les trois systèmes. Nous allons analyser tout d'abord de manière assez précise pour chaque analyseur les résultats d'analyse de la phrase élémentaire utilisée pour la vérification de la bonne installation des systèmes. Ensuite, nous présenterons des remarques globales sur les résultats pour l'ensemble des corpus.

5.5.4 Résultats d'analyse de la phrase d'exemple

Phrase de test

太郎　は　東京駅　で　電車　に　乗る
(Tarô *(prénom)* - [thème] - gare de Tôkyô - [lieu] - train - [destination] - monter)
« Tarô prend le train à la gare de Tôkyô »

Résultats de SAX

Lors de l'analyse de la phrase de test ci-dessus, le système SAX a retourné six arbres syntaxiques n'ayant aucune différence visible (cf. figures 5.11a (voir page suivante), 5.11b page 161 et 5.11c page 162). En effet, cela est dû tout simplement au résultat d'analyse morphologique qui a donné trois réponses pour le mot 太郎 (nom commun *Tarô*, nom de lieu *Tarô* ou prénom *Tarô*), et deux réponses pour 駅 (gare), sans différence visible sur les informations retournées (nom commun pour les deux réponses). Ces différences de résultat d'analyse morphologique ne se reflètent pas dans les arbres d'analyse car elles se situent au niveau de la sous-catégorisation ou au niveau sémantique (polysémie ou homonymie), tandis que l'arbre ne tient compte que des informations syntaxique à partir du niveau de catégorie lexicale.

Il serait peut-être plus intéressant, avec une grammaire aussi simple que celle de SAX, de regrouper les données retournées par l'analyse morphologique en ensembles ayant un caractère commun, pertinent pour l'analyse syntaxique. Étant donné que les différences ne se reflètent pas dans les résultats, il est inutile de réaliser les calculs combinatoires de ce type de données.

La représentation en arborescence montre le concept de base de la grammaire de SAX. En effet, les arbres binaires de SAX représentent l'idée qu'une phrase est considérée comme constituée de deux éléments, complément et prédicat, et qu'un prédicat est constitué lui-même de deux éléments, complément et prédicat, et ainsi de suite.

Résultats de MSLR

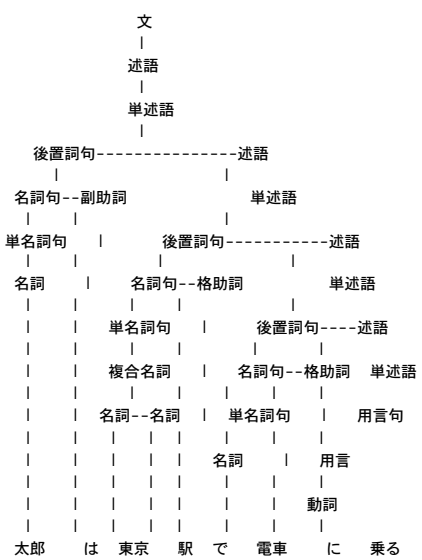
L'analyse morpho-syntaxique de la phrase d'exemple donne comme résultat les dix analyses les plus probables[29]. Mais afin de trouver ces dix réponses plus probables, le système a réalisé le calcul de 216 possibilités.

Cette multiplicité est due non seulement aux ambiguïtés morphologiques – comme pour SAX –, mais aussi aux ambigïtés de niveau syntaxique. En effet, avec la grammaire de MSLR, par exemple, un syntagme constitué d'un substantif et d'une particule *wa*, peut être un *bunsetsu* qui reçoit seulement la qualification *rentai* ou un *bunsetsu* qui peut recevoir à la fois les qualifications *renyô* et *rentai*. Cette présence de double ambiguïté a provoqué

[29] L'analyse est réalisée avec l'option de hiérarchisation des résultats (méthode heuristique), présentée dans la section MSLR 5.3.4 page 115

```
太郎            太郎          名詞        普通名詞
  太郎            太郎          名詞        地名
  太郎            太郎          名詞        人名
は              は            助詞        副助詞
東京            東京          名詞        地名
駅              駅            名詞        普通名詞
  駅              駅            名詞        普通名詞
で              で            助詞        格助詞
電車            電車          名詞        普通名詞
に              に            助詞        格助詞
乗る            乗る          動詞        0       子音動詞ラ行   基本形

形態素解析時間 = 10 msec
```



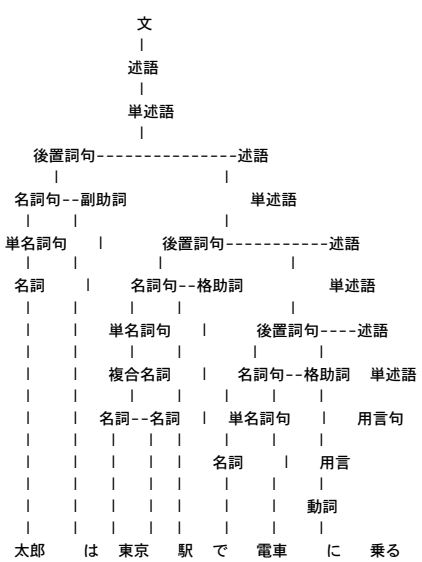
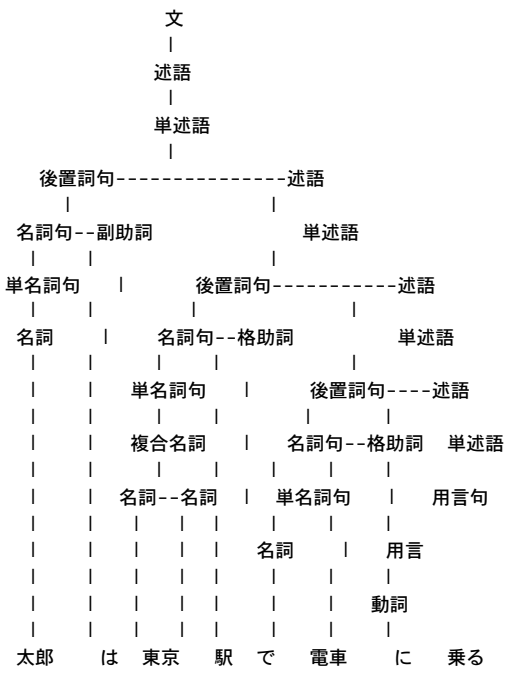


Fig. 5.11a – Résultat de SAX – 1/3



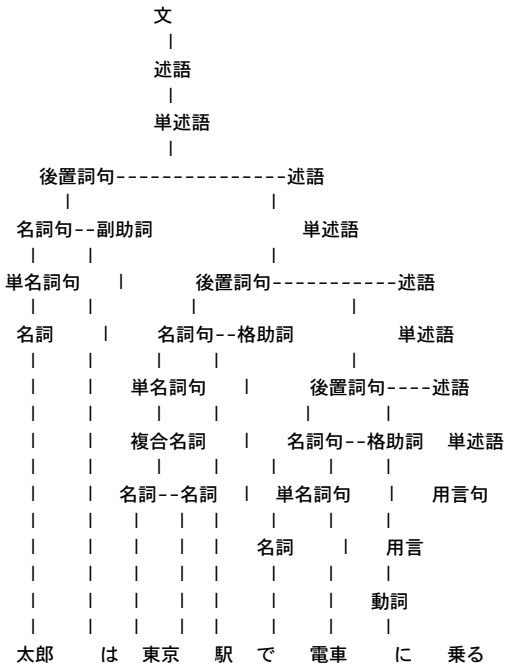
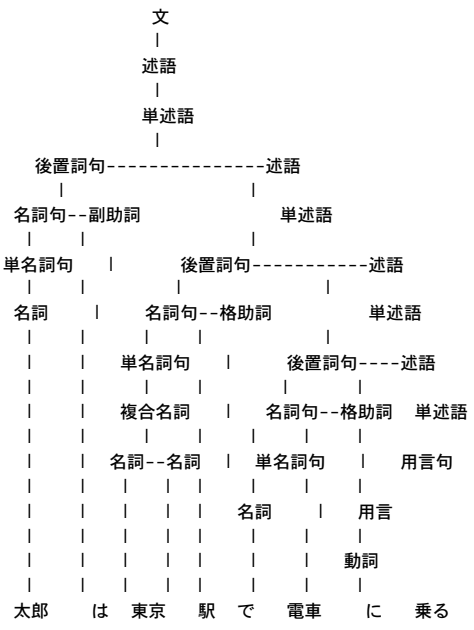
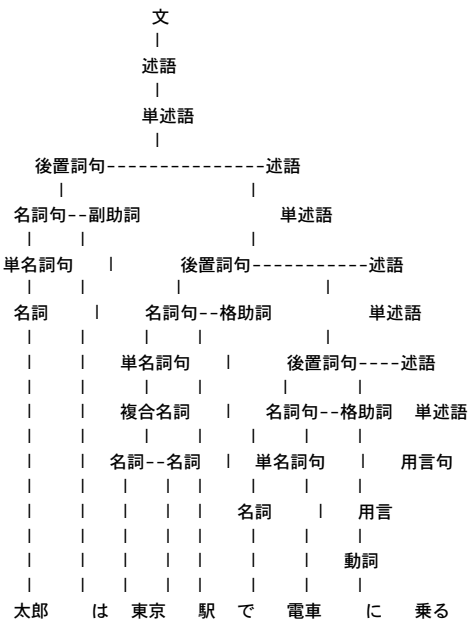


Fig. 5.11b – Résultat de SAX – 2/3






```
execution time (CPU time) = 140 msec.
garbage collection time   = 20 msec.
stack shifting time       = 0 msec.
```

FIG. 5.11c – Résultat de SAX – 3/3

ces calculs importants. Le calcul du système est très rapide mais il suffit, me semble-t-il, que la phrase soit un peu complexe pour provoquer une explosion combinatoire.

Le premier résultat, c'est-à-dire celui considéré comme le plus probable, donne comme analyse morphologique de で « *jodôshi* à la forme converbale », alors que ce mot doit être considéré comme particule *de*, marquant le lieu de l'action. Le premier résultat représente donc non pas la phrase « Tarô prend le train à la gare de Tôkyô », mais quelque chose comme « Tarô est un train et (il) prend (un moyen de transport) à la gare de Tôkyô »[30]. La réponse correcte se trouve aux neuvième et dixième positions avec seulement pour différence le caractère de réception du syntagme thème 太郎は, qui est d'ailleurs une information totalement inutile car dans cette phrase ce syntagme ne reçoit aucune qualification.

Pour la grammaire de MSLR également, une phrase est considérée comme constituée de deux éléments, le syntagme de caractère *renyô* et le syntagme de type *yôgen*. Un syntagme de type *yôgen* est lui-même constitué, tout comme une phrase, de deux éléments.

La figure 5.12 (voir page suivante) montre le résultat retourné par MSLR pour la phrase de test.

Résultat de KNP

Les résultats de KNP présentent deux grandes différences par rapport à ceux des deux autres analyseurs.

Premièrement, KNP ne retourne qu'un seul résultat. La position des concepteurs du système pour le choix des résultats est très nette. Dans le manuel, elle est exprimée comme suit :

> « Nous prenons comme position d'obtenir un seul résultat aussi correct que possible. »

Deuxièmement, l'arbre d'analyse de KNP n'est pas, comme le sont ceux des deux systèmes précédents, de type constituant immédiat, mais de type dépendantiel (cf. section 3.9 page 54).

Pour la phrase de test, KNP propose comme résultat un arbre constitué de la racine étiqueté par le verbe 乗る (monter) et de trois feuilles étiquetées par les syntagmes 太郎は (Tarô [thème]), 東京駅で (gare de Tôkyô [lieu]) et 電車に (train [destination]), toutes les trois étant des descendants directs de la racine.

Cette représentation reflète également la différence de théorie linguistique sur laquelle ils s'appuient : pour KNP, une phrase n'est pas basée sur l'opposition sujet-prédicat, mais elle est constituée d'un élément régissant

[30] Dans une phrase japonaise, n'importe quel complément peut être omis si l'information qu'il apporte est évidente pour les interlocuteurs. Les mots entre parenthèses dans la phrase française sont des éléments absents de la phrase japonaise mais ajoutés comme éléments potentiels pour la construction de la phrase française.

```
### 1 ###
太郎は東京駅で電車に乗る

accept

[<sentence>[<s1>[<bun_用_体_無>[<bun_用体_用_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n_固>[n20,太郎]]]]]
                                                                             [<j_用>[<joshi_用>[<jo_用>[jo62,は]]]]]]
                                                [<bun_用体_用_無>[<b_用体_用_無>[<noun>[<nn>[<n_>[<n_固>[n20,東京]]
                                                                                                  [<n_>[<n>[n10,駅]]]]]]
                                                                                 [<jコ_用>[<jodoushiコ_用>[<jodouコ用>[<jdコ用>[jd39,で]]]]]]]]
                               [<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,電車]]]]]
                                                                           [<j_用>[<joshi_用>[<jo_用>[jo1,に]]]]]]
                                              [<bun_用_体_無>[<b_用_体_無>[<verb止体>[<ve>[ve28,乗]]
                                                                                      [<ev止体>[ev92,る]]]]]]]]]
4.049466e-01
[<sentence>[<s1>[<bun_用_体_無>[<bun_用体_用_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n11,太郎]]]]]
                                                                             [<j_用>[<joshi_用>[<jo_用>[jo62,は]]]]]]
                                                [<bun_用体_用_無>[<b_用体_用_無>[<noun>[<nn>[<n_>[<n_固>[n20,東京]]
                                                                                                  [<n_>[<n>[n10,駅]]]]]]
                                                                                 [<jコ_用>[<jodoushiコ_用>[<jodouコ用>[<jdコ用>[jd39,で]]]]]]]]
                               [<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,電車]]]]]
                                                                           [<j_用>[<joshi_用>[<jo_用>[jo1,に]]]]]]
                                              [<bun_用_体_無>[<b_用_体_無>[<verb止体>[<ve>[ve28,乗]]
                                                                                      [<ev止体>[ev92,る]]]]]]]]]
4.049466e-01
[<sentence>[<s1>[<bun_用_体_無>[<bun_用体_用_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,太郎]]]]]
                                                                             [<j_用>[<joshi_用>[<jo_用>[jo62,は]]]]]]
                                                [<bun_用体_用_無>[<b_用体_用_無>[<noun>[<nn>[<n_>[<n_固>[n20,東京]]
                                                                                                  [<n_>[<n>[n10,駅]]]]]]
                                                                                 [<jコ_用>[<jodoushiコ_用>[<jodouコ用>[<jdコ用>[jd39,で]]]]]]]]
                               [<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,電車]]]]]
                                                                           [<j_用>[<joshi_用>[<jo_用>[jo1,に]]]]]]
                                              [<bun_用_体_無>[<b_用_体_無>[<verb止体>[<ve>[ve28,乗]]
                                                                                      [<ev止体>[ev92,る]]]]]]]]]
4.049466e-01
[<sentence>[<s1>[<bun_用_体_無>[<bun_用体_用_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n_固>[n20,太郎]]]]]
                                                                             [<j_用>[<joshi_用>[<jo_用>[jo62,は]]]]]]
                                                [<bun_用体_用_無>[<b_用体_用_無>[<noun>[<nn>[<n_>[<n>[n10,東京]]
                                                                                                  [<n_>[<n>[n10,駅]]]]]]
                                                                                 [<jコ_用>[<jodoushiコ_用>[<jodouコ用>[<jdコ用>[jd39,で]]]]]]]]
                               [<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,電車]]]]]
                                                                           [<j_用>[<joshi_用>[<jo_用>[jo1,に]]]]]]
                                              [<bun_用_体_無>[<b_用_体_無>[<verb止体>[<ve>[ve28,乗]]
                                                                                      [<ev止体>[ev92,る]]]]]]]]]
4.049466e-01
[<sentence>[<s1>[<bun_用_体_無>[<bun_用体_用_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n11,太郎]]]]]
                                                                             [<j_用>[<joshi_用>[<jo_用>[jo62,は]]]]]]
                                                [<bun_用体_用_無>[<b_用体_用_無>[<noun>[<nn>[<n_>[<n>[n10,東京]]
                                                                                                  [<n_>[<n>[n10,駅]]]]]]
                                                                                 [<jコ_用>[<jodoushiコ_用>[<jodouコ用>[<jdコ用>[jd39,で]]]]]]]]
                               [<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,電車]]]]]
                                                                           [<j_用>[<joshi_用>[<jo_用>[jo1,に]]]]]]
                                              [<bun_用_体_無>[<b_用_体_無>[<verb止体>[<ve>[ve28,乗]]
                                                                                      [<ev止体>[ev92,る]]]]]]]]]
4.049466e-01
[<sentence>[<s1>[<bun_用_体_無>[<bun_用体_用_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,太郎]]]]]
                                                                             [<j_用>[<joshi_用>[<jo_用>[jo62,は]]]]]]
                                                [<bun_用体_用_無>[<b_用体_用_無>[<noun>[<nn>[<n_>[<n>[n10,東京]]
                                                                                                  [<n_>[<n>[n10,駅]]]]]]
                                                                                 [<jコ_用>[<jodoushiコ_用>[<jodouコ用>[<jdコ用>[jd39,で]]]]]]]]
                               [<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,電車]]]]]
                                                                           [<j_用>[<joshi_用>[<jo_用>[jo1,に]]]]]]
                                              [<bun_用_体_無>[<b_用_体_無>[<verb止体>[<ve>[ve28,乗]]
                                                                                      [<ev止体>[ev92,る]]]]]]]]]
4.049466e-01
[<sentence>[<s1>[<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n_固>[n20,太郎]]]]]
                                                           [<j_用>[<joshi_用>[<jo_用>[jo62,は]]]]]]
                               [<bun_用_体_無>[<bun_用体_用_無>[<b_用体_用_無>[<noun>[<nn>[<n_>[<n>[n10,東京]]
                                                                                               [<n_>[<n>[n10,駅]]]]]]
                                                                              [<jコ_用>[<jodoushiコ_用>[<jodouコ用>[<jdコ用>[jd39,で]]]]]]]
                                              [<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,電車]]]]]
                                                                                         [<j_用>[<joshi_用>[<jo_用>[jo1,に]]]]]]
                                                             [<bun_用_体_無>[<b_用_体_無>[<verb止体>[<ve>[ve28,乗]]
                                                                                                    [<ev止体>[ev92,る]]]]]]]]]]
4.049466e-01
[<sentence>[<s1>[<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n11,太郎]]]]]
                                                           [<j_用>[<joshi_用>[<jo_用>[jo62,は]]]]]]
                               [<bun_用_体_無>[<bun_用体_用_無>[<b_用体_用_無>[<noun>[<nn>[<n_>[<n>[n10,東京]]
                                                                                               [<n_>[<n>[n10,駅]]]]]]
                                                                              [<jコ_用>[<jodoushiコ_用>[<jodouコ用>[<jdコ用>[jd39,で]]]]]]]
                                              [<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,電車]]]]]
                                                                                         [<j_用>[<joshi_用>[<jo_用>[jo1,に]]]]]]
                                                             [<bun_用_体_無>[<b_用_体_無>[<verb止体>[<ve>[ve28,乗]]
                                                                                                    [<ev止体>[ev92,る]]]]]]]]]]
4.049466e-01
[<sentence>[<s1>[<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,太郎]]]]]
                                                           [<j_用>[<joshi_用>[<jo_用>[jo62,は]]]]]]
                               [<bun_用_体_無>[<bun_用体_用_無>[<b_用体_用_無>[<noun>[<nn>[<n_>[<n>[n10,東京]]
                                                                                               [<n_>[<n>[n10,駅]]]]]]
                                                                              [<jコ_用>[<jodoushiコ_用>[<jodouコ用>[<jdコ用>[jd39,で]]]]]]]
                                              [<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,電車]]]]]
                                                                                         [<j_用>[<joshi_用>[<jo_用>[jo1,に]]]]]]
                                                             [<bun_用_体_無>[<b_用_体_無>[<verb止体>[<ve>[ve28,乗]]
                                                                                                    [<ev止体>[ev92,る]]]]]]]]]]
4.049466e-01
[<sentence>[<s1>[<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n_固>[n20,太郎]]]]]
                                                           [<j_用>[<joshi_用>[<jo_用>[jo62,は]]]]]]
                               [<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,東京]]
                                                                                           [<n_>[<n>[n10,駅]]]]]]
                                                                          [<j_用>[<joshi_用>[<jo_用>[jo1,で]]]]]]
                                              [<bun_用_体_無>[<bun_体_用_無>[<b_体_用_無>[<noun>[<nn>[<n_>[<n>[n10,電車]]]]]
                                                                                         [<j_用>[<joshi_用>[<jo_用>[jo1,に]]]]]]
                                                             [<bun_用_体_無>[<b_用_体_無>[<verb止体>[<ve>[ve28,乗]]
                                                                                                    [<ev止体>[ev92,る]]]]]]]]]]
4.049466e-01

total 216
CPU time 0.11 sec
```

Fig. 5.12 – Résultat de MSLR

tous les autres éléments (prédicat au sens utilisé par Tesnière), et de ses compléments. D'ailleurs, le système KNP propose la réalisation de l'analyse de la structure argumentale de la phrase, basée probablement sur la grammaire des cas, qui nie l'opposition sujet-prédicat. Cette représentation est, me semble-t-il, beaucoup plus convenable pour représenter la syntaxe japonaise, dans laquelle il est difficile d'accorder un statut particulier au sujet par rapport aux autres compléments.

La figure 5.13 montre le résultat retourné par KNP pour la phrase de test.

FIG. 5.13 – Résultat de KNP

5.5.5 Résultats d'analyse du corpus par SAX

SAX n'a donné aucun résultat pour les 50 phrases étudiées.

Cet échec total est premièrement dû à une mauvaise correspondance entre le programme et la grammaire. En effet, en plus des *joshi* (particules) pour lesquels nous avons modifié le code lors de l'installation du système, d'autres catégories posent le même problème, à savoir : alors qu'à partir du résultat de l'analyse morphologique le système conserve comme donnée la catégorie lexicale de chaque mot, dans la grammaire, ces mêmes catégories sont traitées avec leur sous-catégorie lexicale. Cependant, d'autres catégories ne possédant pas de sous-catégories, il est impossible de résoudre ce problème en modifiant tout simplement le code pour que le système conserve toujours la sous-catégorie lexicale.

De plus, il manque dans la grammaire une règle élémentaire permettant d'analyser les syntagmes nominaux constitués de deux substantifs reliés par la particule *no* (*grosso modo* équivalent à la préposition « de » en français).

En réglant ces deux problèmes ne serait-ce que de manière provisoire, et en ajoutant une règle de grammaire de manière à traiter les blocs de mots inconnus comme des substantifs, SAX a réussi à analyser au moins trois phrases simples du corpus. Nous avons tout de même, rien qu'avec cette petite amélioration de la grammaire, rencontré une erreur dûe à une insuf-

fisance de mémoire. Cela montre bien la faiblesse de l'algorithme, évoquée par les créateurs eux-même dans l'article [23], où une solution à ce problème est proposée.

L'algorithme étant très intéressant, il est dommage de n'avoir pu évaluer correctement cet analyseur. Il serait certainement intéressant de le réétudier avec une grammaire correcte et l'amélioration présentée dans la section 5.2.9 page 107. D'ailleurs, d'après l'article [23], une grammaire HPSG du japonais est en cours de développement dans un laboratoire du NAIST.

5.5.6 Résultats d'analyse du corpus par MSLR

Analysons les résultats ouvrage par ouvrage. Nous allons consacrer une partie à chaque ouvrage, avec à chaque fois, la présentation dans un tableau des résultats d'analyse de MSLR, du nombre d'arbres générés et du positionnement du résultat correct parmi les dix arbres retournés, avec éventuellement des informations sur le corpus.

L'analyse est réalisée avec les options de « Traitement des mots inconus » et « Hiérarchisation des résultats par méthode heuristique ». Le nombre d'affichage de résultats est la valeur par défaut, à savoir 10. Zéro (0) pour le positionnement signifie qu'il y a pas de résultat qui semble correct, et le symbole « ? » pour le nombre d'arbres générés signifie que le nombre total dépassant la valeur maximale d'un `long int` (en langage C), le système n'a pas pu le compter. Le résultat 0/0 signifie que l'analyse a échoué. Pour toute analyse ayant duré plus de vingt minutes, nous avons décidé d'interrompre les calculs et de considérer le résultat comme inconnu. Il est alors représenté par ?/ ?.

Ouvrage 1 : Initiation aux langages formels et aux automates

N° de phrase	1	2	3	4	5	6	7	8	9	10
Nb de mots anglais	4	2	0	0	2	0	0	1	0	0
Nb de symboles	0	0	0	1	0	1	9	0	0	4
Nb d'arbres	0	0	18 321 408	320	0	3 584	1 198	0	2 688	3
Rés. correct	0	0	0	1	0	9 et 10	0	0	9 et 10	2

TAB. 5.7 – Initiation aux langages formels et aux automates

On constate d'après le tableau 5.7 que l'analyse de MSLR échoue lorsque la phrase contient des mots anglais. En revanche, le système réalise bien l'analyse des phrases comprenant des formules mathématiques. En effet, comme nous en avions fait l'hypothèse, il considère les formules mathématiques comme un bloc de mots inconnus.

Ouvrage 2 : Grammaire II d'Iwanami

N° de phrase	1	2	3	4	5	6	7	8	9	10
Nb d'arbres	?	318 944 000	?	0	0	?	0	?	?	0
Rés. correct	0	0	0	0	0	0	0	0	0	0

TAB. 5.8 – Grammaire II d'Iwanami

Comme on peut le constater dans le tableau 5.8 , le système n'a retourné aucun résultat qui semble correct. Lorsque la phrase contient plusieurs propositions subordonnées relatives, le système semble avoir du mal à trouver le prédicat correct de la proposition principale, en considérant comme prédicat le *yôgen* le plus proche, qui est en fait le prédicat d'une proposition subordonnée.

Parmi quatre échecs de l'analyse, deux cas ont échoué visiblement au niveau morphologique. En effet, l'analyse s'est arrêtée après l'occurrence du même terme, un qualificatif démonstratif, *konoyôna*. Dans un autre cas, elle s'est arrêtée après la conjonction, *saraniwa*. Soit le système ne supporte pas la coordination des propositions, soit le problème se trouve dans ce cas également au niveau morphologique.

Ouvrage 3 : Brevet

N° de phrase	1	2	3	4	5	6	7	8	9	10
Nb d'arbres	?	54 864	0	0	0	0	0	0	?	?
Rés. correct	?	0	0	0	0	0	0	0	?	?

TAB. 5.9 – Brevet

Le système n'a retourné aucun résultat qui semble correct.

Parmi six échecs de l'analyse, le dernier (la phrase 8) est dû au terme *konoyôna*, comme dans les phrases de l'ouvrage précédent. Les cinq autres ont échoué à l'occurrence du verbe 含浸させる, qui ne figure sans doute pas dans le dictionnaire. Ce verbe est un verbe composé de forme substantif + verbe *suru*, très utilisé. Le substantif constituant ce verbe est lui-même un mot composé de deux idéogrammes, 含 (contenir) et 浸 (plonger), traduisant ainsi le sens « imprégner ». Ce terme composé est rare au point qu'il est quasiment impossible de le trouver ailleurs que dans ce brevet. Il ne figure donc évidemment pas dans le dictionnaire. L'échec d'analyse de ce verbe composé montre que, bien que le système arrive à traiter les formules mathématiques en les considérant comme des mots inconnus, il ne peut pas traiter les verbes composés, si le composant substantif ne se trouve pas sous une forme présente telle quelle dans le dictionnaire. Ce qui est un défaut

grave car la production de substantifs de cette manière est assez courante, en particulier dans les documents techniques tels que les brevets.

Ouvrage 4 : Roman de Murakami

N° de phrase	1	2	3	4	5	6	7	8	9	10
Nb d'arbres	96 102 048	?	2 240	0	0	37 134 720	0	64 512	9 216	0
Rés. correct	0	0	0	0	0	0	0	0	0	0

Tab. 5.10 – Roman de Murakami

Pour ces phrases également, le système n'a retourné aucun résultat qui semble correct.

Parmi quatre échecs de l'analyse, l'un (la phrase 5) est probablement dû à l'analyse du niveau morphologique. L'analyse de cette phrase s'est arrêtée avec la séquence *subeki*. C'est une forme plus ancienne de *surubeki*, dont la règle de construction n'est sans doute pas prise en compte par le système.

Ouvrage 5 : Article du quotidien Asahi

N° de phrase	1	2	3	4	5	6	7	8	9	10
Nb d'arbres	?	0	?	0	?	0	?	?	?	0
Rés. correct	0	0	0	0	0	0	0	?	0	0

Tab. 5.11 – Article du quotidien Asahi

Pour ces phrases également, le système n'a retourné aucun résultat qui semble correct.

Parmi quatre échecs de l'analyse, deux phrases se terminent par un substantif non suivi d'une copule. C'est une façon de terminer une phrase assez courante en japonais, appelée 体言止め(*taigen-dome*, terminaison en *taigen*). Il est vraisemblable que la grammaire du système ne tient pas compte de cette manière de terminer les phrases.

Remarques générales

Tout d'abord, la présentation de résultats était assez difficile à lire. On aurait préféré, même au sacrifice de certaines informations, une forme plus agréable à lire.

En commençant par l'analyse morphologique, les possibilités se multiplient tellement que le nombre d'arbres générés est extrêmement important. Dans ce cas, il est question de trouver la bonne analyse parmi ces nombreuses possibilités. D'après ce que nous avons pu observer, la méthode heuristique de hiérarchisation des résultats n'est pas très efficace. Mais plutôt que de

trouver une méthode plus fiable pour choisir le résultat le plus probable, ne serait-il pas plus intéressant de désambiguïser dès que cela est possible ? Calculer 96 102 048 possibilités n'est pas forcément toujours utile.

Le système analyse relativement bien les propositions simples. Afin de généraliser cette capacité, il serait peut-être intéressant de concevoir un système qui détecte les propositions constituant la phrase et qui utilise MSLR comme module pour le traitement des propositions simples.

L'atout de MSLR est situé dans le traitement des formules mathématiques. Ce mécanisme assez simple apporte déjà un résultat très intéressant.

5.5.7 Résultats d'analyse du corpus par KNP

Nous passons maintenant à l'analyse des résultats de KNP, ouvrage par ouvrage.

KNP a donné un résultat pour les cinquante phrases analysées.

Dans les tableaux présentant les résultats d'analyse de KNP, ◯ signifie que le résultat semblait correct, et × signifie l'obtention d'un résultat erroné. Pour les résultats « presque » corrects, c'est-à-dire où l'erreur se limite seulement à une unité lexicale telle qu'un adverbe ou une conjonction, le symble △ est utilisé.

Ouvrage 1 : Initiation aux langages formels et aux automates

N° de phrase	1	2	3	4	5	6	7	8	9	10
Résultat	×	×	×	◯	◯	◯	×	△	×	×

Tab. 5.12 – Initiation aux langages formels et aux automates

KNP analyse bien la relation entre la proposition principale et la proposition subordonnée. Mais la coordination entre les propositions parait moins bien interprétée.

KNP ne sait pas bien traiter non plus les formules mathématiques. Le résultat montre que le système a analysé le formule $G = (V, E)$, ce qui est probablement inutile, voire néfaste. Il semble plus intéressant de les traiter comme MSLR en un seul bloc sans s'attacher au contenu lui-même.

Ouvrage 2 : Grammaire II d'Iwanami

N° de phrase	1	2	3	4	5	6	7	8	9	10
Résultat	×	×	×	◯	◯	×	×	◯	×	△

Tab. 5.13 – Grammaire II

La détection des structures de coordination fonctionne relativement bien. Mais, dans la phrase 7, la coordination des deux propositions de forme interrogative, pourtant bien marquée par la répétition de la particule *ka*, ～か、～かを n'a pas été bien interprétée.

Ouvrage 3 : Brevet

N° de phrase	1	2	3	4	5	6	7	8	9	10
Résultat	×	◯	×	×	◯	×	◯	×	×	×

Tab. 5.14 – Brevet

Tout comme pour MSLR, le verbe composé 含浸させる a perturbé le système KNP.

La troisième phrase du texte original contenant une faute, l'omission d'une particule *no*, l'analyseur n'a pas réussi à segmenter correctement cette partie. Ce qui n'est pas particulièrement étonnant.

L'analyse de ces résultats nous a permis de nous rendre compte d'un défaut intéressant de KNP. Tout en donnant des résultats satisfaisants pour la coordination de petits syntagmes, le système ne connaît probablement pas les structures de coordination dans lesquelles les éléments coordonnés ont non seulement le même régissant mais aussi le même complément. En d'autres termes, pour la phrase :

j'ai acheté des abricots et des pêches de Méditerranée

le système peut détecter la relation de coordination entre *abricots* et *pêches*, mais *de Méditerranée* est attaché uniquement à l'unité la plus proche, *pêches* en l'occurrence. Les résultats des phrases 4 et 5 sont erronés pour cette raison.

Ouvrage 4 : Roman de Murakami

N° de phrase	1	2	3	4	5	6	7	8	9	10
Résultat	×	×	×	◯	×	◯	×	◯	×	◯

Tab. 5.15 – Roman de Murakami

Deux grands constats sont apparus suite à l'analyse de ces phrases.

Premièrement, les propositions de condition marquées par la particule *ba* semblent ne pas être correctement interprétées.

Deuxièmement, le *keisiki-meishi* (substantif formel) *no* étant inclut dans le même syntagme que son qualifiant, la représentation syntaxique produite par KNP paraît trop différente de la structure syntaxique de la phrase. Les manipulations de KNP pour la création de certains syntagmes particuliers

posent, me semble-t-il, un certain nombre de problèmes. Nous en parlerons plus précisément dans la partie ci-dessous sur les remarques générales.

Ouvrage 5 : Article du quotidien Asahi

N° de phrase	1	2	3	4	5	6	7	8	9	10
Résultat	△	×	×	○	×	○	×	×	○	○

Tab. 5.16 – Article du quotidien Asahi

KNP supporte bien l'omission de la copule lors de la terminaison en substantif.

La structure de coordination des verbes, marquée par ～たり～たり, de la phrase 2 n'a pas été détectée. Ce modèle appartenant aux modèles principaux des structures de coordination, il est dommage, KNP ayant comme atout la détection des structures de coordination, de ne pas pouvoir analyser correctement cette structure.

Remarques générales

La représentation du résultat de type dépendantiel extrêmement simple de KNP est une forme très agréable à lire bien qu'elle ne soit pas suffisante si des informations plus détaillées sont nécessaires.

KNP interprète bien le schéma global des relations entre les syntagmes. En particulier, l'analyse de certaines relations de coordination entre syntagmes et de relations entre la proposition principale et la proposition subordonnée relative est un travail considérable de KNP.

Néanmoins, les détails des résultats entraînent de nombreuses interrogations. Est-il légitime d'inclure *keishiki-meishi*, substantif formel *no*, dans le même syntagme que son qualifiant en sachant que cela entraîne la génération de deux arbres complètement différents pour les phrases 昨日買った本をなくした (j'ai perdu le livre que j'ai acheté hier) et 昨日買ったのをなくした (j'ai perdu ce que j'ai acheté hier) ? De même, est-il juste de construire l'arbre syntaxique en laissant dans le dernier syntagme coordonné la particule indiquant pourtant la fonction syntaxique de tous les syntagmes coordonnés ? Toute question de cette nature fait revenir en fait au problème du syntagme *bunsetsu*, déjà évoqué dans le chapitre 1. Le découpage en syntagmes *bunsetsu* ne correspondant pas à la structure sémantique des phrases, l'arbre syntaxique basé sur ces syntagmes ne peut pas représenter les relations exactes entre les éléments constituant la phrase.

De plus, pour certaines opérations, plus d'études purement linguistiques semblent nécessaires. Par exemple, KNP définit comme règle de construction des mots composés, qu'un substantif de temps suivi d'un nom commun en

kanji forme un substantif composé avec ce dernier. Cette règle, assez curieuse sur le plan linguistique, empêche d'analyser correctement la fonction principale des substantifs de temps qu'est l'emploi adverbial. Par exemple, la phrase 今朝電車に乗った (j'ai pris un train ce matin) est analysée comme constituée de deux syntagmes 今朝電車に et 乗った (monter dans une voiture, un train ou un avion), le premier syntagme n'ayant aucun sens. En effet, 今朝 (ce matin), substantif de temps, suivi d'un nom commun 電車 (train), a formé un substantif composé selon la règle décrite précédemment.

La distinction entre les phrases constituées de deux propositions, subordonnée et principale, et celles qui sont composées de deux propositions coordonnées est également un point important à étudier et à bien définir.

Conclusion

Nous avons présenté, à travers ce document, les différentes connaissances de base et les constituants de l'analyse syntaxique automatique du japonais.

Nous nous sommes tout d'abord intéressés aux notions de linguistique japonaise et à des généralités sur l'analyse syntaxique, nous permettant ainsi d'acquérir les connaissances de base indispensables à la compréhension de notre sujet principal.

Cet aperçu de la linguistique japonaise nous a permis de cerner les grandes lignes des problèmes de ce domaine, notamment celui de la définition des unités linguistiques. Ces unités linguistiques constituant les unités élémentaires du TAL, ce problème est directement lié à nos travaux et sa prise en considération sérieuse, impérative.

L'étude générale de l'analyse syntaxique et la présentation des principaux algorithmes nous ont permis de découvrir l'existence de nombreuses méthodes d'analyse syntaxique, ainsi que le profil algorithmique de l'analyse syntaxique. Ce caractère, obtenu par séparation totale de la grammaire et de l'algorithme, apporte la possibilité de généralisation des opérations. Ainsi, un analyseur d'une langue donnée peut réaliser l'analyse syntaxique d'une autre langue sans modification du mécanisme même du système, mais seulement grâce à un changement de grammaire. Il est néanmoins justifié de se poser la question de l'intérêt réel de cette possibilité de généralisation des systèmes : nous connaissons très bien aujourd'hui l'existence de points communs à toute langue, mais leur diversité est d'autant plus connue. Les divergences, parfois même surprenantes, entre les langues peuvent remettre en cause cette recherche de généralisation d'un traitement pour des langues très différentes.

L'étude des principaux analyseurs du japonais existants à ce jour a constitué un des points essentiels du présent mémoire. Ces études nous ont montré, ne serait-ce que partiellement, la réalité des techniques actuelles, leurs points forts et leurs faiblesses.

Nous n'avons malheureusement pas pu évaluer correctement le système SAX. Cependant, nous avons pu non seulement constater sa principale fai-

blesse, à savoir la nécessité d'un important espace de stockage des données, mais également étudier une solution possible à ce problème, proposée par les créateurs du système. Il serait certainement intéressant de réétudier cet outil d'analyse syntaxique avec une grammaire correcte et l'amélioration proposée pour le stockage des données.

Le système MSLR analyse lui relativement bien les propositions simples. Mais son grand défaut réside dans la prise en compte d'ambiguïtés en quantité astronomique. Ce constat nous a appris l'importance de la désambiguïsation : il est peut-être même indispensable de procéder parfois à un choix plutôt que de conserver simplement toutes les possibilités.

KNP quant à lui interprète bien le schéma global des relations entre les syntagmes. Néanmoins, les détails des résultats entraînent de nombreuses interrogations notamment du point de vue linguistique. Certaines manipulations sur les règles morpho-syntaxiques semblent tenir compte uniquement de phénomènes limités à des unités données, permettant certes le traitement de cas particuliers, mais empêchant une analyse correcte de la fonction générale de ces unités. L'idée principale sur laquelle repose cet analyseur est intéressante, mais il est dommage que ses créateurs n'aient pas pris plus en compte les travaux des linguistes.

De façon générale, les obstacles que rencontrent ces analyseurs semblent provenir plus d'un manque d'éléments linguistiques que d'une question purement informatique. La prise en compte, même faible, des travaux des linguistes apporterait probablement une amélioration considérable.

Cet ensemble d'études a premièrement remis en cause un point sur lequel nous n'avons pas suffisamment discuté : l'objectif de l'analyse syntaxique.

En effet, sans se définir exactement un objectif, il est impossible de savoir ce que nous devons produire comme résultat. Il est très important afin de fixer un objectif, de définir tout d'abord ce qu'est la phrase pour le TAL, objet de l'analyse syntaxique. Cela permettra de comprendre ce qu'il est possible de retourner comme résultat. Selon Minoru WATANABE, la phrase est constitué de 叙述 (*jojutsu*, dictum) et de 陳述 (*chinjutsu*, modus). Il définit 陳述 (*chinjutsu*) comme une fonction syntaxique qui détermine, en prenant le contenu de 叙述 (*jojutsu*) – exprimant une pensée ou un fait – comme matière, le rapport entre le contenu de 叙述 (*jojutsu*) et le sujet parlant. Le résultat d'analyse doit-il représenter l'ensemble de ces structures ? Ou bien, considère-t-on seulement le dictum comme l'élément de niveau syntaxique ? Selon le choix retenu, nous devons traiter différemment la notion de thème.

La présente étude nous a également permis de réfléchir à certains points concrets pour la réalisation d'un analyseur syntaxique du japonais.

Sur le plan linguistique, il est avant tout impératif d'examiner différentes théories existantes, de manière à trouver ou à élaborer la grammaire la plus adaptée au TAL.

De plus, il serait sûrement intéressant d'introduire le concept de l'opposition thème-rhème. L'opposition thème-rhème (ou *topic-comment*) est souvent traitée comme un élément du niveau énonciatif, mais pour une langue comme le japonais, dans laquelle le thème est bien marqué par une particule, il me semble erroné d'ignorer sa présence au niveau syntaxique. Cette introduction de la notion de thème apporterait sans doute des avantages pour les applications telles que l'extraction des informations ou le résumé automatique. Néanmoins, pour la traduction automatique, ce concept serait peut-être néfaste avec les langues cibles ne possédant pas l'opposition thème-rhème du moins au niveau de surface, même s'il reste sûrement avantageux lorsqu'il s'agit de la traduction entre deux langues de nature semblable telles que le japonais et le coréen.

Enfin, l'analyse des relations entre les propositions constituant la phrase semble être également un point à développer. Afin de concevoir un système tenant compte des indications linguistiques, les travaux réalisés par Fujio MINAMI et Yasuo KITAHARA ([43], [44] et [30]) devraient nous fournir un fondement très intéressant.

Sur le plan algorithmique, il serait sûrement intéressant de concevoir un algorithme d'analyse procédant à un balayage du texte source non pas de gauche à droite mais de droite à gauche ou même bi-directionnel. En effet, il est possible de poser comme hypothèse que pour les langues ayant le prédicat à la fin de la phrase telles que le japonais, il est plus facile de déterminer l'étendue de la proposition en partant de cette fin de la phrase où se trouve le prédicat, possédant en général des informations sur les éléments constituant la proposition. Mais avant de commencer l'analyse par la fin de phrase, il faudrait réaliser l'analyse de tous les thèmes situés en tête de phrase, d'où la nécessité d'avoir un algorithme à balayage bi-directionnel. Les thèmes sont généralement, comme présenté par Charles N. Li et Sandra A. Thompson dans [36], mis en avant dans une phrase.

Afin d'illustrer cet algorithme, imaginons la construction d'un arbre syntaxique. Avec l'introduction de la notion de thème, la première branche gauche pourrait représenter le premier thème de la phrase et la branche droite le sous-arbre représentant le rhème correspondant. On construirait ainsi d'abord toutes les branches gauches (en nombre égal au nombre de thèmes), puis les sous-arbres représentant le rhème grâce à une analyse débutant par la fin de phrase.

Il est difficile de dire si cette hypothèse présente un réel bénéfice pour la réalisation d'un analyseur syntaxique du japonais, mais il serait en tout état de cause intéressant, si cela n'a pas encore été réalisé, de comparer cet algorithme avec ceux de type plus classique, étudiés dans le présent document.

La place qu'occupent les moyens informatiques dans notre vie ne cesse de croître. Toutefois, la plupart des gens ne peuvent pas encore profiter de la moitié des possibilités dont ils disposent réellement. Ce gâchis est partiel-

lement dû au manque de logiciels efficaces, capables de traiter correctement nos langues. Les sujets de recherche dans le domaine du TAL sont réellement loin d'être épuisés.

Bibliographie

[1] Alfred V. AHO, Ravi SETHI, and Jeffrey D. ULLMAN. *Compilateurs – Principes, techniques et outils – Cours et exercices*. Dunod, Paris, 2000.

[2] Alfred V. AHO and Jeffrey D. ULLMAN. *The theory of Parsing, Translation and Compiling*, volume I : Parsing. Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[3] André ARNOLD and Irène GUESSARIAN. *Mathématiques pour l'informatique*. Masson, Paris, troisième edition, 1997.

[4] Ch. BALLY. *Linguistique générale et linguistique française*. Francke, Berne, quatrième edition, 1965.

[5] Philippe BLACHE. *Les grammaires de propriétés*. Hermes Science, Paris, 2001.

[6] Leonard BLOOMFIELD. *le Langage*. Payot, Paris, 1970.

[7] Ivan BRATKO. *PROLOG Programming for Artificial Intelligence*. Addison-Wesley, Harlow, third edition, 2001.

[8] Wallace L. CHAFE. Givenness, contrastiveness, definiteness, subjects, topics, and point of view. In *Subject and topic*, pages 25 – 55. Academic Press Inc., New York, 1976.

[9] Noam CHOMSKY. *Aspects de la théorie syntaxique*. Seuil, Paris, 1971.

[10] Communications Research Laboratory. *EDR Electronic Japanese Word Dictionary*. http ://www.jsa.co.jp/EDR.

[11] Thomas CORMEN, Charles LEISERSON, and Ronald RIVEST. *Introduction à l'algorithmique*. Dunod, Paris, 1994.

[12] Izuru SHINMURA Dir. 広辞苑 *[Kôjien]*. Iwanami, Tokyo, fifth edition, 1999. Ressource électronique.

[13] Jean-Marie PIERREL Dir. *Ingénierie des langues*. Informatique et Systèmes d'Information. HERMES, Paris, 2000.

[14] Susumu NAGARA Dir. 日本語教育能力検定試験傾向と対策 *[Tendance et prévention pour l'examen de performance d'enseignement du japonais]*. Babel press, Tokyo, 1991.

[15] Jean DUBOIS, Mathée GIACOMO, Louis GUESPIN, Christiane MARCELLESI, Jean-Baptiste MARCELLESI, and Jean-Pierre MÉVEL. *Dictionnaire de linguistique et des sciences du langage*. Larousse, Paris, 1999.

[16] Oswald DUCROT and Jean-Marie SCHAEFFER Dir. *Nouveau dictionnaire encyclopédique des sciences du langage*. Seuil, Paris, 1995.

[17] Masakazu FUJIO and Yuji MATSUMOTO. Japanese dependency structure analysis based on statistics. *IPSJ SIG Notes*, NL - 117(12) :83 – 90, january 1997.

[18] Gerald GAZDAR and Chris MELLISH. *Natural Language Processing in Prolog*. Addison-Wesley, Wokingham, 1989.

[19] Dick GRUNE, Henri E. BAL, Ceriel J.H. JACOBS, and Koen G. Langendoen. *Compilateurs – Cours et exercices corrigés*. Dunod, Paris, 2002.

[20] Shinkichi HASHIMOTO. 国語法要説. Meiji Shoin, Tokyo, 1934.

[21] Oki HAYASHI, Haruhiko KINDAICHI, and Takeshi SHIBATA Dir. *An Encyclopaedia of the Japanese Language*. Taishukan Shoten, Tôkyô, 1988.

[22] Peter HELLWIG. A course in cooking – natural language parsers. Juillet 1999.

[23] Osamu IMAICHI, Masakazu FUJIO, Youhei YAMANE, Takashi MIYATA, and Yuji MATSUMOTO. An efficient parsing method using dependency information and its application to robust parsing. *IPSJ SIG Notes*, NL - 121(8) :53 – 60, september 1997.

[24] Kiyoshi ISHIHATA. アルゴリズムとデータ構造 *[Algorithmes et structures de données]*, volume 3 of 岩波講座日本語ソフトウェア科学 *[Cours d'Iwanami : science du logiciel]*. Iwanami Shoten, Tôkyô, 2002.

[25] Hiroyasu ITSUI, Masahiro UEKI, Takeyuki AIKAWA, Kenji SUGIYAMA, and Hozumi TANAKA. A method of unkown word detection in mslr parser. *IPSJ SIG Notes*, NL - 121(5) :31 – 37, september 1997.

[26] Yasunori KANDA. 日本語情報処理の方向. In 日本語情報処理シンポジウム, July 1978. 17 - 20.

[27] Yasunori KANDA. 日本語情報処理のために. *Bit*, 14(14) :1762 – 1767, December 1982.

[28] Brian W. KERNIGHAN and Dennis M. RITCHIE. *Le langage C*. Masson, Paris, deuxième edition, 1992. Troisième tirage.

[29] Margaret KING, editor. *Parsing Natural Language*. Academic Press, London, 1983.

[30] Yasuo KITAHARA. 日本語助動詞の研究. Taishukan, Tôkyô, 1981.

[31] S.Y. KURODA. Le jugement catégorique et le jugement thétique ; exemples tirés de la syntaxe japonaise. *Langages*, 30 :81 – 110, 1973.

[32] Sadao KUROHASHI. 結構やるな、knp. *IPSJ Magazine*, 41(11) :1215 – 1220, november 2000.

[33] Sadao KUROHASHI and Makoto NAGAO. A method for analyzing conjunctive structures in japanese. *IPSJ SIG Notes*, NL - 86(2) :1 – 8, november 1991.

[34] Sadao KUROHASHI and Makoto NAGAO. A method for analyzing conjunctive structures in japanese. *IPSJ Journal*, 33(8) :1022 – 1031, august 1992.

[35] Sadao KUROHASHI and Makoto NAGAO. *Japanese Morphological Analysis System JUMAN*. Kyôto University, 1998.

[36] Charles N. LI and Sandra A. THOMPSON. Subject and topic : a new typology of language. In *Subject and topic*, pages 457 – 491. Academic Press Inc., New York, 1976.

[37] Hiroshi MAKINO and Makoto KIZAWA. Automatic segmentation for transformation of kana into kanji. *IPSJ Journal*, 20(4) :337 – 345, july 1979.

[38] Takashi MASUOKA and Yukinori TAKUBO. 基礎日本語文法. Kurishio Shuppan, Tokyo, 1992.

[39] Yuji MATSUMOTO, Taro KAGEYAMA, Masaaki NAGATA, Hirofumi SAITO, and Takenobu TOKUNAGA. 単語と辞書 *[Mots et dictionnaires]*, volume 3 of *Linguistic Sciences*. Iwanami Shoten, 1997.

[40] Yuji MATSUMOTO, Akira KITAUCHI, Tatsuo YAMASHITA, Yoshitaka HIRANO, Hiroshi MATSUDA, Kazuma TAKAOKA, and Masayuki ASAHARA. *Morphological Analysis System Chasen 2.2.9*. Nara Institute of Science and Technology, 2002.

[41] Daizaburo MATSUSHITA. 改撰標準日本文法. Kigensha, Tôkyô, 1928.

[42] Akira MIKAMI. 象は鼻が長い. Kuroshio Shuppan, Tôkyô, 1960.

[43] Fujio MINAMI. 現代日本語の構造. Taishukan, Tôkyô, 1974.

[44] Fujio MINAMI. 現代日本語文法の輪郭. Taishukan, Tôkyô, 1993.

[45] Susumu ÔNO and Takeshi SHIBATA Dir. 文法 *I [Grammaire I]*, volume 6 of 岩波講座日本語 *[Cours d'Iwanami : japonais]*. Iwanami Shoten, Tôkyô, 1976.

[46] Susumu ÔNO and Takeshi SHIBATA Dir. 文法 II *[Grammaire II]*, volume 7 of 岩波講座日本語 *[Cours d'Iwanami : japonais]*. Iwanami Shoten, Tôkyô, 1977.

[47] Fernando C.N. PEREIRA and Stuart M. SHIEBER. *Prolog and Natural-Language Analysis*. CSLI, Stanford, 1987.

[48] Leon STERLING and Ekud SHAPIRO. *L'art de Prolog*. Masson, Paris, 1990.

[49] Hozumi TANAKA. 自然言語解析の基礎 *[Base de l'analyse linguistique du japonais]*. Sangyô Tosho, Tôkyô, third edition, 1989.

[50] Hozumi TANAKA. 自然言語処理 – 基礎と応用 [natural language processing and its applications]. *Denshi jôhô tsûshin Gakkai*, 1999.

[51] Tanaka & Tokunaga Lab. Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology. *Morphological and Syntactic LR (MSLR) parser tool kit (Ver.1.0)*, november 1999.

[52] Lucien TESNIÈRE. *Éléments de Syntaxe Structurale*. KLINCKSIECK, Paris, deuxième edition, 1988. Cinquième tirage.

[53] Takayuki TOMITA and Kazuko SANADA. 表記 *[Systèmes d'écriture]*. The Japan Foundation, Tokyo, 1988.

[54] Minoru WATANABE. 国語構文論. Haniwa Shobo, Tôkyô, 1971.

[55] Reinhard WILHELM and Dieter MAURER. *Les compilateurs – Théorie, construction, génération*. Masson, Paris, 1994.

[56] Terry WINOGRAD. *Language as a Cognitive Process*, volume I : Syntax. Addison-Wesley, 1983.

[57] Takao YAMADA. 日本文法論. Hobunkan, Tôkyô, 1908.

[58] Takao YAMADA. 日本文法学概論. Hobunkan, Tôkyô, 1936.

[59] Kenji YOSHIMURA, Tooru HITAKA, and Sho YOSHIDA. 日本語文の形態素解析における最長一致法と文節数最小法について. *IPSJ SIG Notes*, NL - 30(7) :1 – 6, march 1982.

[60] Kenji YOSHIMURA, Tooru HITAKA, and Sho YOSHIDA. Morphological analysis of non-marked-off japanese sentences by the least bunsetsu's number method. *IPSJ Journal*, 24(1) :40 – 46, january 1983.