Laboratoire Systèmes et Transports **Université de Technologie de Belfort Montbéliard**

Méta - grammaire et calcul sémantique pour les Grammaires d'Arbres Adjoints

MÉMOIRE

présenté et soutenu publiquement le 30 juin 2003

pour l'obtention du

D.E.A. de l'université de Franche Comté – Besançon

(Option Informatique Automatique Productique)

par

Yannick PARMENTIER

Composition du jury

Président :	Abderrafiaa KOUKAM	*Professeur des Universités*
Examinateurs :	Abdellah EL MOUDNI	*Professeur des Universités*
	Vincent HILAIRE	*Maître de Conférences*
	Yassine RUICHEK	*Maître de Conférences*
Encadrants :	Claire GARDENT	
	Bertrand GAIFFE	

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Table des matières

Avant-propos

Avant d'entrer dans le vif du sujet, il peut être utile de resituer le cadre de ce stage.
Ce stage correspond d'une part à mon projet de fin d'études intervenant dans ma formation d'ingénieur, formation dispensée par l'Université de Technologie de Belfort Montbéliard. D'autre part, ce stage entre en compte dans la préparation du Diplôme D'Etudes Approfondies mention Informatique - Automatique - Productique de l'Université de Franche - Comté.
L'objectif de ce stage est donc double : permettre une mise en application dans un environnement professionnel des connaissances acquises au cours des études, et fournir une initiation à la recherche dans l'optique d'une poursuite en thèse.
Ce stage a été effectué au Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA), au sein de l'équipe «Langue et Dialogue».
Le LORIA est né d'une tradition de recherche forte à Nancy, succédant au Centre de Recherche en Informatique de Nancy (CRIN) ; il regroupe 5 partenaires au sein de l'unité mixte de recherche UMR 7503 : le Centre National de Recherche Scientifique (CNRS), l'Institut National de Recherche en Informatique et Automatique (INRIA), l'Institut National Polytechnique de Lorraine (INPL), l'Université Henri Poincaré - Nancy 1 et l'Université Nancy 2. Les missions du LORIA sont multiples, elles vont de la conduite de recherches fondamentales et appliquées, à la réalisation de systèmes expérimentaux en passant par le transfert des connaissances (via des échanges internationaux notamment), la formation par la recherche et la valorisation des résultats de la recherche. Plus de 300 personnes travaillent dans le laboratoire, réparties entre 21 équipes de recherche et 7 équipes d'aide à la recherche. Les thématiques de recherche actuelles sont les suivantes :

- Bioinformatique et applications à la génomique,
- Calculs, réseaux et graphismes à hautes performances,
- Ingénierie des langues, du document et de l'information scientifique, technique et culturelle,
- Qualité et sûreté des logiciels,
- Télé-opérations et assistants intelligents.

Les relations industrielles représentent une activité importante pour le LORIA (près de 200 contrats en cours fin 1999), d'ailleurs cette même année a été fondé le club Loriatech, regroupant les partenaires du laboratoire, et ce en vue de faciliter la veille technlogique des entreprises et de permettre des partenariats sur des appels d'offres français ou européens.
En outre, le LORIA conçoit et diffuse des logiciels, 6 logiciels ainsi qu'une

marque ont été déposés en 1998. Précisons enfin l'existence depuis 1999 d'un incubateur offrant une aide à la création d'entreprise (2 entreprises ont été créées en 2000).

L'équipe «Langue et Dialogue» prend place naturellement au sein de la thématique de l'ingénierie des langues et vise à l'introduction du langage dans l'interface Homme Machine. L'activité de l'équipe s'articule autour des axes suivants :

- Etude des mécanismes fondamentaux de la communication en langue naturelle seule ou accompagnée d'une désignation gestuelle (dialogue multimodal), cet axe allie les disciplines de l'informatique, de la logique et de la linguistique,
- Réalisation de systèmes de dialogue effectifs dans le cadre notamment de collaborations industrielles,
- Définition d'outils et de méthodes génériques permettant d'étudier de façon fine des situations de dialogues réels, issus de la transcription d'expériences de simulation ou d'observations directes.

L'équipe se compose de 22 membres permanents (Chercheurs, Ingénieurs experts, Enseignants-chercheurs, Assistants de projets), de 7 chercheurs doctorants (dont certains en co-tutelle de thèse), de 2 chercheurs post-doctorants, de 6 chercheurs associés et de 2 étudiants en stage de DEA. Les collaborations de l'équipe sont nombreuses tant avec l'industrie qu'avec les institutions régionales, nationales (Actions de Recherche Concertées INRIA GenI et RLT notamment) et internationales (projet européen MIAMM par exemple). Dans ce cadre divers logiciels ont été développés, dont InDiGen (générateur de texte en langue naturelle), CURT (construction sémantique et inférence pour les langues naturelles), MGC (éditeur/compilateur de méta-grammaires), etc.

Le chapitre d'introduction qui suit, présente le sujet de ce stage et sa problématique.

Introduction

Le cadre général de ce travail correspond à la génération de langue naturelle et plus particulièrement à la phase dite de «Réalisation». Le principe du procédé de Réalisation est le suivant : à partir d'une représentation sémantique de l'information à transmettre, produire le (ou les) textes associés à cette représentation via une grammaire. A l'issue de cette phase, nous disposons du texte en langue naturelle, ainsi que de l'arbre syntaxique correspondant.
Plusieurs approches sont envisageables pour réaliser cela, celle que nous considérerons dans ce document est celle de l'Action de Recherche Concertée INRIA «GenI», dans laquelle une grammaire associe à une chaîne de mots :

- une structure syntaxique (1) et
- une représentation sémantique (2).

A l'heure actuelle, il n'existe pas de grammaire de type Grammaire d'Arbres Adjoints (Tree Adjoining Grammar ou TAG) pour le français qui intègre (2). L'objectif des travaux en cours réside donc dans la production d'une grammaire TAG pour le français à large couverture qui permette l'association d'une représentation sémantique aux phrases générées.
Le moyen employé pour réaliser cette tâche consiste en l'utilisation d'une part, d'une **méta-grammaire**, terme désignant un ensemble de règles permettant de générer des règles de production d'une grammaire, et d'autre part d'un **compilateur** qui est un outil informatique de génération de grammaires à partir d'une méta-grammaire.
C'est ici qu'intervient ce stage de DEA, dont l'objectif est l'étude de la génération automatique d'une grammaire TAG à portée sémantique, ce qui suppose d'aborder des domaines tels que la syntaxe ou encore les logiques de descriptions.
Dans un premier temps, il sera donné une vue d'ensemble du formalisme TAG, de son fonctionnement et de ses intérêts linguistiques et formels. Nous introduirons alors le procédé de génération automatique d'une grammaire TAG, sa motivation, les approches existantes, leurs avantages et inconvénients. Ce qui nous amènera à la présentation de deux compilateurs de méta-grammaire développés actuellement au LORIA, le compilateur MGC et le résolveur de descriptions colorées. Nous étudierons notamment dans ces parties les problèmes liés à l'intégration d'une représentation sémantique. Enfin nous conclurons par un bilan du travail réalisé et présenterons quelques extensions envisageables.

Chapitre 1

Présentation des Grammaires d'Arbres Adjoints

1.1 Définition d'une grammaire TAG

De manière formelle, une grammaire d'arbres adjoints G est définie par le 5-uplet suivant :

$$G = (NT, T, P, I, A)$$

où :

- NT représente l'ensemble fini des symboles non-terminaux (que nous appelerons également catégories),
- T l'ensemble fini des symboles terminaux (les mots du lexique dans notre cas), $NT \cap T = \emptyset$,
- P est le symbole distingué ou axiome,
- I correspond à l'ensemble fini des arbres initiaux,
- A correspond à l'ensemble fini des arbres auxiliaires[1],

5-uplet auquel on associe des opérations de combinaison entre éléments de $I \cup A$ (i.e. entre arbres), opérations appelées *adjonction* et *substitution*.
Une grammaire TAG est un système de réécriture d'arbres, à la différence des grammaires hors-contexte[2] (qui procèdent par réécriture de chaînes).
Dans les sections qui suivent, nous allons définir plus en détails ce que sont les composants et opérations de combinaison cités ci-dessus. Ensuite nous évoquerons les propriétés d'une grammaire TAG, enfin nous explorerons un procédé d'interfaçage entre représentation syntaxique et sémantique dans une grammaire TAG.

1.1.1 Composants d'une grammaire TAG

L'analyse avec une grammaire TAG fait intervenir les éléments suivants :

Des arbres initiaux : Ils sont notés α et se caractérisent par des noeuds intérieurs étiquetés par des symboles non-terminaux, et des noeuds feuilles qui

[1] Arbres initiaux et arbres auxiliaires forment ce que l'on appelle les arbres élémentaires.
[2] appelées aussi CFG, de l'anglais « Context Free Grammars ».

sont soit étiquetés par des terminaux, soit étiquetés par des non-terminaux et marqués pour une substitution (symbole $\downarrow$).

Des arbres auxiliaires : Ils sont notés β. Leur spécificité réside dans le fait qu'ils disposent parmi leurs noeuds feuilles d'un noeud étiqueté par le même symbole que la racine de l'arbre, ce noeud est alors appelé "noeud pied" et est marqué par le symbole *.

Des arbres dérivés : Ils correspondent à des arbres issus d'une ou plusieurs combinaisons d'arbres (les opérations de combinaison autorisées sont présentées section 1.1.2).

Des arbres de dérivation : A chaque arbre dérivé correspond un arbre de dérivation représentant comment les différents arbres élémentaires ont été combinés (voir section 1.1.5).

De plus, nous ferons le choix d'une grammaire TAG lexicalisée, c'est-à-dire que chaque arbre élémentaire (initial ou auxiliaire) sera associé à un item lexical (qui sera appelé «ancre» de l'arbre), cet item peut correspondre à une seule feuille ou à plusieurs (on parle alors d'ancre multi-composants).

1.1.2 Opérations de combinaison d'arbres

Pour composer entre eux des arbres d'une grammaire TAG, on dispose de deux opérations : **l'adjonction**[3], et **la substitution**.
L'adjonction peut être vue comme l' «insertion» d'un arbre à l'intérieur d'un autre. Précisons que les seuls arbres pouvant être insérés ainsi sont les arbres auxiliaires. Si l'on considère un arbre auxiliaire α_1 de noeud pied X, et un arbre élémentaire γ dont un noeud est étiqueté par X (racine ou noeud intérieur), alors l'arbre dérivé de l'adjonction de α_1 dans γ au noeud X, est l'arbre γ' tel que le noeud X de γ où a eu lieu l'adjonction est remplacé par le noeud racine X de α_1, α_1 est alors placé sous ce noeud, et le sous-arbre qui était dominé par X dans γ est déplacé sous le noeud pied de l'arbre α_1 inséré. On peut voir un exemple d'adjonction sur la figure 1.1.

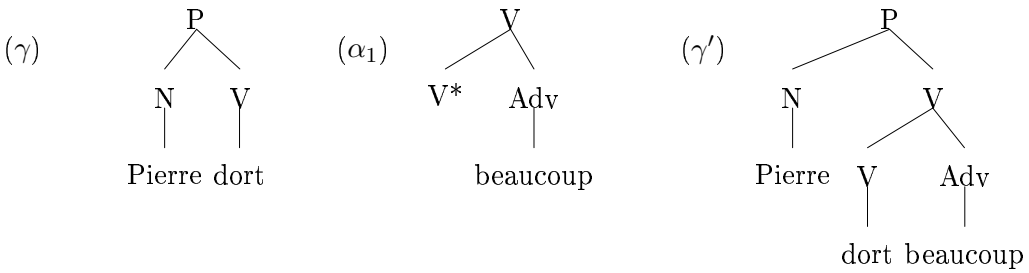


FIG. 1.1 – Exemple d'adjonction.

Cette figure représente l'adjonction de l'arbre auxiliaire de l'adverbe "beaucoup" au noeud V de l'arbre dérivé "Pierre dort".

L'opération de substitution correspond au liage d'un arbre initial β_1 à une feuille

[3] à l'origine de la théorie, c'était la seule opération de combinaison autorisée.

d'un arbre δ, feuille qui doit être marquée par $\downarrow$ et étiquetée par le même symbole que la racine de β_1. Un exemple de substitution est donné figure 1.2.

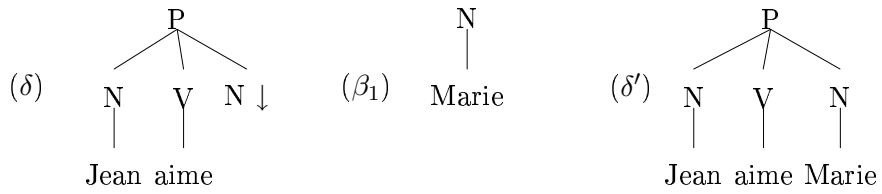


FIG. 1.2 – Exemple de substitution.

Nous avons substitué l'arbre initial "Marie" au noeud N $\downarrow$ de l'arbre dérivé "Jean aime N".

Il convient de noter que l'arbre résultant d'une opération de combinaison est du même type que l'arbre de départ, c'est-à-dire que par exemple si l'on substitue un arbre initial à une feuille d'un arbre auxiliaire, le résultat est un arbre auxiliaire.
La question qui se pose alors, une fois que nous savons comment combiner les arbres entre eux, est : « Quels sont les arbres dont nous disposons au départ ? ». La section suivante va répondre à cette question.

1.1.3 Principes de formation des arbres élémentaires

Dans [Abe93], l'auteur énonce des règles de formation des arbres élémentaires reposant sur des considérations linguistiques. Ces règles sont les suivantes :

Principe d'ancrage lexical : tout arbre élémentaire a au moins une tête lexicale non-vide, ce qui reprend notre hypothèse de grammaire TAG lexicalisée,

Principe de coocurrence prédicat - arguments : tout prédicat contient dans sa structure élémentaire au moins un noeud pour chacun des arguments qu'il sous-catégorise[4] (par exemple l'arbre élémentaire du verbe « aimer » disposera au moins de deux noeuds, l'un pour le sujet et l'autre pour le complément d'objet direct),

Principe de consistence sémantique : tout arbre élémentaire a un correspondant sémantique non-vide, c'est-à-dire que des arbres vides de sens tels que ceux ancrés uniquement par une conjonction comme « qui » seront interdits,

Principe de non-compositionalité : un arbre élémentaire n'a qu'un seul correspondant sémantique, on limite la taille des arbres élémentaires de telle sorte qu'ils ne puissent représenter qu'une seule unité sémantique.

Ces principes contraignent la génération des structures élémentaires et permettent au formalisme TAG de présenter des avantages certains par rapport

[4] C'est-à-dire pour les arguments du prédicat logique correspondant : ici, aime(x,y).

aux formalismes utilisés jusqu'alors, tels que les grammaires hors-contexte (voir section 1.2).

1.1.4 Grammaires TAG et structures de traits

Les structures de traits[5] ont été introduites initialement par le formalisme des grammaires fonctionnelles d'unification. A l'origine, elles proviennent du traitement des phénomènes d'accord. Plutôt que d'écrire des grammaires dont les symboles non-terminaux sont du type « NomCommunMasculinSingulier » ou « DéterminantMasculinSingulier », on a choisi de se limiter à des catégories simples (Nom, Déterminant), de leur associer des traits d'accord (nombre, genre) et de contraindre l'accord par des équations.
L'application des structures de traits aux grammaires TAG remonte à [VSJ88] et a donné lieu à un nouveau type de grammaire : les **grammaires FTAG** (Feature structures based Tree Adjoining Grammars). En TAG, ces traits sont associés aux noeuds, et sont tous à valeur atomique. Ils peuvent donner une information morphologique, syntaxique ou encore sémantique. Les traits associés à un noeud dépendent de la catégorie du noeud, pour le français nous avons, entre autres, les traits suivants : <**det**> qui distingue les noms propres des noms communs, <**mode**> pour indiquer le mode du verbe, etc.
Les traits interviennent lors de la combinaison d'arbres, en effet ils conditionnent la substitution ou l'adjonction à un noeud. Les structures de traits des noeuds impactés par l'opération de combinaison doivent pouvoir s'unifier. Si l'unification échoue, la combinaison est interdite[6].
On associe aux noeuds d'un arbre TAG deux structures de traits : une structure **top** et une structure **bottom**. Lors de l'adjonction à un noeud X la structure de traits *top* de ce noeud doit s'unifier avec la structure *top* de la racine de l'arbre auxilaire et la structure *bottom* avec la structure *bottom* du pied de l'arbre auxiliaire (voir figure 1.3).

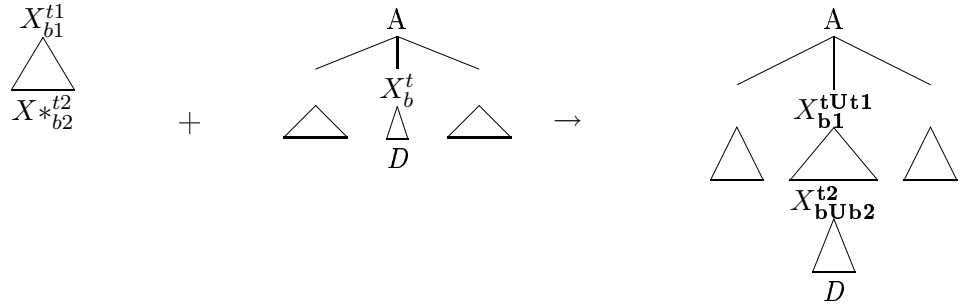


FIG. 1.3 – Structures de traits et adjonction d'arbres.

Pour une substitution, seules les structures *top* de la feuille concernée et du noeud racine de l'arbre initial doivent s'unifier.
Notons qu'à la fin du processus de réécriture d'arbres (qui peut être composé

[5] « feature structure » en anglais, un trait est un couple *attribut-valeur*.
[6] On retrouve l'idée des contraintes d'adjonction énoncées dans [KJ85].

d'une ou de plusieurs combinaisons d'arbres successives), les traits *top* et *bottom* doivent être en mesure de s'unifier à chaque noeud[7].
Le trait *top* représente les liens entre un noeud et les noeuds le dominant, et le trait *bottom* les liens entre un noeud et les noeuds qu'il domine. [Abe93] classe les traits morpho-syntaxiques en 3 groupes : les **traits d'accord** (propagés via la structure *top*, trait <num> par exemple), **de tête** (propagés également via la structure *top*, trait <mode> par exemple), et **de pied** (propagés par la structure *bottom*, trait <qu> par exemple).
Les structures de traits permettent de représenter des informations linguistiques diverses dans un formalisme défini (processus d'unification). La figure 1.4 illustre le rôle joué par les structures de traits dans la sous-catégorisation. L'arbre ancré par le verbe «vouloir» impose une complétive à l'infinitif, ce qui se traduit par la présence du trait <mode> (et non pas par une catégorie spécifique étiquetant le noeud d'adresse 2.2).

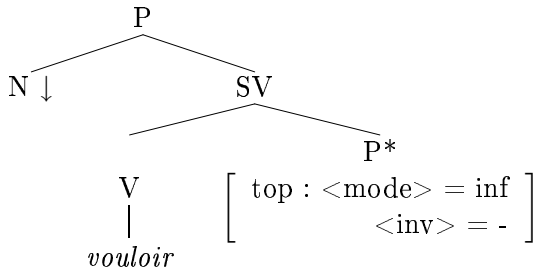


FIG. 1.4 – Structure de traits et sous-catégorisation.

Le fait d'incorporer les contraintes de combinaison entre arbres dans des structures de traits permet une instanciation dynamique de ces contraintes, alors qu'à l'origine du formalisme TAG, ces contraintes devaient être spécifiées au préalable. Les structures de traits permettent donc une économie d'arbres.
Elles permettent en outre de contraindre de manière significative les combinaisons d'arbres et de ce fait présentent un intérêt computationnel non - négligeable (lors de l'analyse syntaxique notamment).

1.1.5 Exemple de grammaire TAG

Intéressons nous à la grammaire TAG de la figure 1.5[8].

Cette grammaire permet de générer les deux phrases "Jean aime Marie" et "Jean donne une pomme à Marie". Nous allons donner respectivement les arbres dérivés correspondant à chacune de ces phrases ainsi que leur arbre de dérivation.
Notons que cette grammaire est constituée de 3 arbres initiaux complets[9] "Marie", "Jean", et "pomme", ainsi que les arbres initiaux associés aux verbes

[7]On remarquera que la présence dans un arbre élémentaire d'un noeud dont les structures de traits *top* et *bottom* diffèrent représente une contrainte d'adjonction obligatoire.

[8]Dans un souci de simplification, nous n'utilisons pas les structures de traits dans cet exemple, le lecteur pourra néanmoins voir un exemple de leur emploi à la section 1.3.

[9]Ils ne contiennent pas de feuilles à substituer.

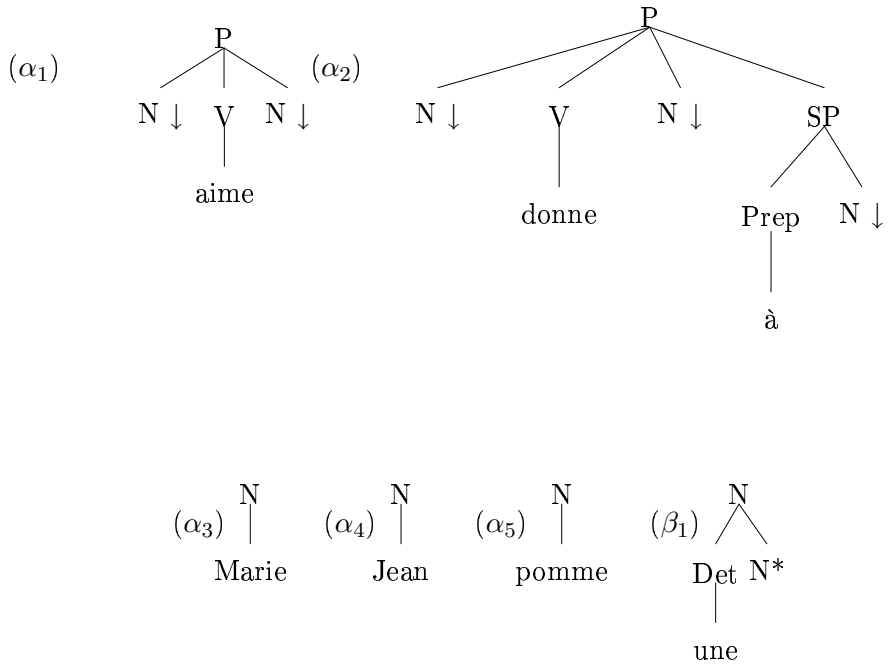


FIG. 1.5 – Exemple de Grammaire TAG.

"aime" et "donne à", et un arbre auxiliaire correspondant au déterminant "une". La phrase "Jean aime Marie" est représentée par l'arbre ε_1 et le processus de construction est représenté par l'arbre η_1 de la figure 1.6.

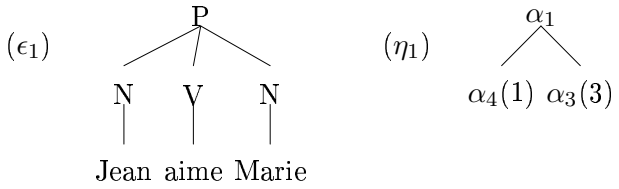


FIG. 1.6 – Arbre dérivé et arbre de dérivation - exemple 1.

L'arbre dérivé correspond à la substitution respectivement de l'arbre "Jean" au noeud de catégorie N d'adresse 1 et de l'arbre "Marie" au noeud de catégorie N d'adresse 3, ce qui se lit sur l'arbre de dérivation. On remarque que les noeuds de l'arbre de dérivation correspondent aux arbres utilisés, que les arcs en trait continu représentent une substitution (des traits pointillés sont utilisés pour représenter l'adjonction), enfin le numéro entre parenthèses aux différents noeuds représente l'adresse de Gorn de la feuille où a eu lieu la substitution.
De même l'arbre dérivé, ainsi que l'arbre de dérivation, de "Jean donne une pomme à Marie" sont représentés figure 1.7 et notés respectivement ϵ_2 et η_2.

L'arbre de dérivation η_2 se lit comme suit : on adjoint l'arbre de "une" à la racine de l'arbre "pomme" (arc en pointillé), et on substitue respectivement les

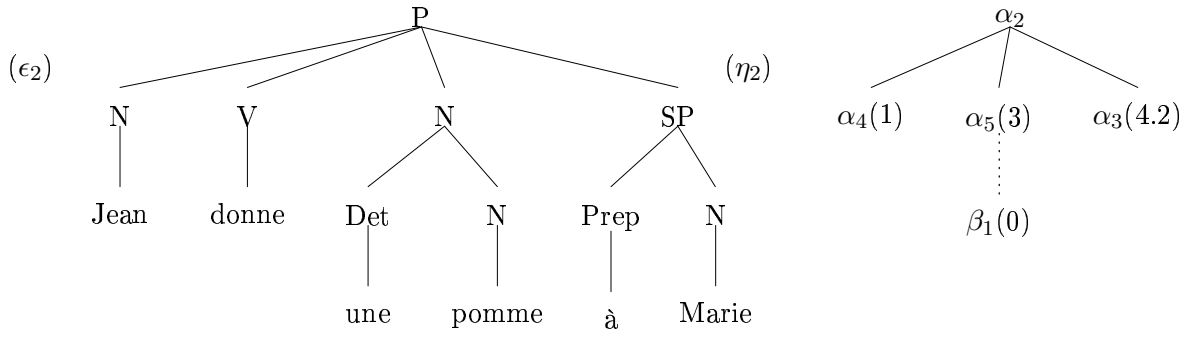


FIG. 1.7 – Arbre dérivé et arbre de dérivation - exemple 2.

arbres de "Jean", "une pomme" et "Marie" aux adresses 1, 3 et 4.2 dans l'arbre de "donne à".
On remarque que les branches de l'arbre de dérivation ne sont pas ordonnées, en effet on peut substituer l'arbre de "Jean" avant celui de "Marie" et vice versa. L'arbre de dérivation est proche de l'arbre des dépendances, il donne une représentation du lien entre un prédicat et ses arguments. Il est considéré comme outil de base de l'interprétation sémantique, nous reviendrons là dessus à la section 1.3.

1.2 Quelques propriétés linguistiques et formelles d'une grammaire TAG

Le formalisme TAG, d'abord étudié pour ses caractéristiques mathématiques, s'est révélé particulièrement bien adapté au traitement du langage. Il dispose en effet de propriétés linguistiques intéressantes parmi lesquelles, nous pouvons citer :

Un domaine de localité étendu : Contrairement aux grammaires hors-contexte, qui sont représentables par des arbres de profondeur 1, les grammaires TAG sont constituées de règles de production de profondeur supérieure, ce qui permet, en accord avec le principe de coocurrence prédicat - arguments (PCPA), de disposer de structures regroupant un prédicat directement avec ses arguments. Les dépendances sont donc représentées sur le même arbre élémentaire.

Une récursion du domaine de dépendances : Cette propriété découle du même PCPA, les arbres élémentaires TAG couvrent le domaine sur lequel existent des dépendances telles que l'accord, ou la sous-catégorisation. L'adjonction permet alors de conserver ces dépendances dans le même arbre élémentaire, même si elles sont éloignées dans la phrase (par exemple lors de l'adjonction d'une structure récursive telle que l'arbre élémentaire « croire que», la phrase *Qui crois tu que Jean crois que Pierre aime ?* est obtenue par adjonction de l'arbre « croire que» à un noeud P de l'arbre de « Qui crois-tu que Pierre aime ? »).

Une grammaire « légèrement » sensible au contexte : La classe des langages générés par des grammaires TAG lexicalisées (ou LTAG) contient la classe des langages hors - contextes, et est contenue dans la classe des langages contextuels[10].

Un pouvoir génératif « fort » : Cette propriété est liée à la précédente. Grâce à l'opération d'adjonction, une grammaire LTAG peut générer de manière forte le même langage qu'une grammaire hors - contexte. Par génération « forte », nous entendons que la grammaire LTAG permet non seulement de générer les mêmes phrases que la grammaire hors-contexte, mais également les mêmes structures dérivées (plus précisément, une grammaire TAG permet de lexicaliser une grammaire hors-contexte).

Existence d'un automate de reconnaissance : Pour chaque langage d'une grammaire TAG, il existe un automate de type Embedded Push - Down qui reconnaît ce langage de manière non - équivoque.

Parsing en un temps polynomial : Les langages d'arbres adjoints sont analysables syntaxiquement en un temps polynomial, la complexité des algorithmes d'analyse est en O^6.

Le lecteur trouvera des informations complémentaires sur les propriétés des grammaires TAG dans [KJ85] et [JS97].

1.3 Grammaires TAG et représentation sémantique

Tout d'abord précisons ce que l'on entend par représentation sémantique. La représentation sémantique correspond à une formule logique associée à une expression, et donne des informations sur le sens de celle-ci. Disposer d'une représentation du sens d'une phrase est un enjeu important, par exemple dans un but d'extraction d'information. Depuis Montague (1974), les travaux menés en sémantique visent la construction d'une représentation sémantique combinée au procédé d'élaboration de la structure syntaxique. Alors que de nombreuses avancées ont été faites dans ce sens pour les grammaires syntagmatiques guidées par les têtes (HPSG), les grammaires catégorielles d'unification (UCG) ou encore les grammaires fonctionelles lexicales (LFG), il s'avère que les grammaires d'arbres adjoints ne disposent pas encore d'un mécanisme de calcul sémantique adéquat. Les théories émises pour les grammaires TAG s'appuient généralement sur l'arbre de dérivation, car ce dernier a l'avantage de représenter le lien entre un prédicat et ses arguments. Cependant certains problèmes tels que la représentation des quantificateurs subsistent. L'approche suivie ici est celle développée dans [GK03] employant la *sémantique plate* comme support de représentation sémantique et basant le calcul sémantique sur l'arbre dérivé (plutôt que sur l'arbre de dérivation).

[10] Langages dont les règles de production sont de la forme aAb → aBAb avec A symbole non-terminal.

1.3.1 Sémantique plate

La sémantique plate utilise une logique fondée sur celle présentée dans [Bos95], augmentée de variables d'unification[11]. Cette logique présente certains avantages tels que la réduction des ambiguïtés de portée (via un procédé de sous - spécification), ou la composition des formules élémentaires lors de dérivations (via l'utilisation du procédé d'unification de traits).
Le principe de la sémantique plate appliquée aux grammaires TAG consiste à associer des formules logiques du premier ordre aux arbres élémentaires, formules qui vont être composées[12] lors de la génération de phrases par combinaison d'arbres.
Les objets utilisés dans cette représentation sont les suivants :

- des constantes individuelles (notées sous formes de chaînes de caractères),
- des constantes individuelles d'unification (notées a, b, c, ...),
- des variables individuelles d'unification (notées x_i),
- des constantes de label d'unification (notées l_i),
- des variables de label d'unification (notées s_i),
- des constantes de portées (notées h_i),
- des relations n-aires prenant en paramètre des constantes individuelles[13], des variables individuelles ou des constantes de portée,
- et des relations de portée (notées $\geq$).

Les formules utilisées sont de deux types suivant qu'elles comportent des variables d'unification ou non : formules d'unification et formules saturées.
Formellement, la définition d'un formule d'unification en logique L_U est la suivante :

Définition 1 *Soient l une constante ou variable de label, h une constante de portée, i_1,...,i_n des variables ou constantes individuelles ou constantes de portée, et R^n une relation n-aire, alors :*

1. $l : R^n(i_1, ..., i_n)$ *est une formule* L_U.
2. $h \geq l$ *est une formule* L_U.
3. ϕ,ψ *est une formule* L_U *ssi* ϕ *est une formule* L_U *et* ψ *est une formule* L_U.
4. *ce sont les seules formules* L_U.

Expression en langue naturelle	**Formule L_U associée**
un	$l : \exists(x, h_1, h_2)\ ,\ h_1 \geq s_1\ ,\ h_2 \geq s_2$
chien	$l : Chien(x)$
un chien aboie	$l_0 : \exists(x_1, h_1, h_2)\ ,\ h_1 \geq l_1, h_2 \geq l_2\ ,\ l_1 : Chien(x_1)\ ,\ l_2 : Aboyer(x_1)$

FIG. 1.8 – Exemples de représentation sémantique en logique L_U

[11] D'où le terme généralement utilisé de « logique d'unification L_U ».
[12] Cette composition correspond en pratique à une conjonction des formules élémentaires.
[13] Ces constantes peuvent être d'unification ou non.

Un intérêt fort de cette logique est la sous-spécification des relations de portée permettant de représenter les ambiguïtés sous forme « factorisée». En L_U, cette sous - spécification est représentée à l'aide de l'opérateur $\geq$ symbolisant une relation de portée au sens large entre deux prédicats.

Définition 2 *Soit ϕ une formule saturée (i.e. sans variable d'unification). Alors la relation de portée entre une constante de portée et une variable de label $\geq_\phi$ est définie par :*
$\forall k, k', k'' \in L_\phi \cup H_\phi$[14]

1. $k \geq_\phi k$.
2. $k \geq_\phi k'$ *si* $k \geq k'$ *est dans* ϕ.
3. $k \geq_\phi k''$ *si* $k \geq_\phi k'$ *et* $k' \geq_\phi k''$.
4. *si* $\exists k : \tau$ *avec* $\tau(.., k', ..)$ *alors* $k \geq_\phi k'$ *et* $\neg(k' \geq_\phi k)$.
5. *si* $\exists R^n(.., k, .., k', ..) \in \phi$ *alors* $\neg(k \geq_\phi k')$ *et* $\neg(k' \geq_\phi k)$.
6. *ce sont les seules représentations de* $\geq_\phi$.

La propriété 4 exprime le fait que, si une variable de label apparaît comme argument d'une relation, alors cette variable a une portée moindre que la constante de label étiquetant la relation en question.
A ce stade nous n'avons pas abordé la question de la résolution des ambiguïtés contenues dans une formule sous - spécifiée. Cela passe par l'emploi d'une fonction injective $P : H_\phi \rightarrow L_\phi$. Notons ϕ' la formule L_U résultante du remplacement dans ϕ de toutes les constantes de portée $k \in H_\phi$ par $P(k)$. La fonction P réalise un branchement (plugging en anglais) entre une constante de portée et une variable de label. La résolution d'une ambiguïté de portée correspond à la détermination de tous les branchements possibles pour une formule saturée ϕ :

Définition 3 *P est un branchement possible pour une formule saturée ϕ* ***ssi*** *$\forall k, k' \in L_\phi$: si $k \geq_{\phi'} k'$ alors soit $k = k'$, soit $\neg(k' \geq_{\phi'} k)$.*

Afin de clarifier ces notions, regardons l'exemple (1) ayant pour formule L_U associée (2) :

(1) Tout homme aime une femme.

(2) $l_0 : \forall(x_1, h_1, h_2)$, $h_1 \geq l_1, h_2 \geq l_2$, $l_1 : Homme(x_1), l_2 : Aimer(x_1, x_2)$, $l_3 : \exists(x_2, h_3, h_4)$, $h_3 \geq l_4$, $h_4 \geq l_2$, $l_4 : Femme(x_2)$

Il s'en suit les branchements acceptables suivants :

1. $\{h_1 \mapsto l_1$, $h_2 \mapsto l_2$, $h_3 \mapsto l_4$, $h_4 \mapsto l_0\}$, ce qui correspond à la formule $l_0 : \forall(x_1, l_1, l_2)$, $l_1 : Homme(x_1), l_2 : Aimer(x_1, x_2)$, $l_3 : \exists(x_2, l_4, l_0)$, $l_4 : Femme(x_2)$ (i.e. tous les hommes aiment la même femme),
2. $\{h_1 \mapsto l_1$, $h_2 \mapsto l_3$, $h_3 \mapsto l_4$, $h_4 \mapsto l_2\}$, ce qui correspond à la formule $l_0 : \forall(x_1, l_1, l_3)$, $l_1 : Homme(x_1), l_2 : Aimer(x_1, x_2)$, $l_3 : \exists(x_2, l_4, l_2)$, $l_4 : Femme(x_2)$ (i.e. tous les hommes aiment une femme, pas forcément la même).

[14] L_ϕ et H_ϕ représentent respectivement l'ensemble des variables de label et celui des constantes de portée.

Dans la section qui suit, nous allons aborder l'intégration de la sémantique plate dans les grammaires TAG, et comment via le processus d'unification nous pouvons opérer la composition sémantique lors de la génération de phrases.

1.3.2 Interface syntaxe / sémantique

Rappelons que dans une grammaire TAG, les phrases sont produites par combinaison d'arbres élémentaires, il existe 2 types de combinaisons possibles : la substitution et l'adjonction. De plus des structures de traits *top* et *bottom* garnissent les noeuds des arbres, ces structures conditionnent la combinaison d'arbres via l'unification[15] des traits des noeuds impliqués (voir 1.1.4).

Comme vu en 1.3.1, l'idée de base dans notre approche est d'associer une formule L_U à chaque arbre élémentaire, et d'utiliser dans cette formule des variables d'unification qui sont coindexées avec les traits *top* et *bottom* de certains noeuds. Plus précisément, la structure de trait *top* contient l'information qui doit être filtrée dans le cas d'une adjonction, et la structure *bottom* l'information locale au noeud. Le procédé d'unification permet ainsi, lors de la combinaison d'arbres, de mettre à jour les variables de la formule L_U. A la fin du processus de réécriture, nous disposons d'une part d'un arbre complet (i.e. qui ne contient pas de noeuds marqués $\downarrow$ ou $*$), et d'autre part d'une formule logique L_U composée correspondant à la conjonction des formules des arbres élémentaires dont certaines variables sont partagées.

Regardons cela de plus près au travers de l'exemple utilisé précédemment : « Tout homme aime une femme ». Pour réaliser l'interfaçage syntaxe / sémantique dans une grammaire TAG couvrant cette phrase, nous allons utiliser des traits spécifiques nommés *index* et *label*.

La première étape consiste en la décoration de certains noeuds dans les arbres élémentaires de notre grammaire (voir figure 1.9[16]).

La seconde étape consiste en la combinaison des arbres élémentaires, la mise à jour des variables des formules L_U est alors prise en charge par les mécanismes de coindexation de traits et d'unification. L'arbre TAG correspondant à notre exemple « Tout homme aime une femme » et sa formule L_U sont donnés figure 1.10.

Lors de l'adjonction de l'arbre auxiliaire d'un déterminant à l'arbre initial d'un nom pour produire l'arbre d'un groupe nominal, il se produit une unification entre traits *index et label* du noeud pied de l'arbre du quantificateur et du noeud racine de l'arbre du nom. En conséquence, les variables de label s_i de la formule de sémantique plate sont unifiées avec les constantes l_i, de même les variables individuelles x_i sont unifiées entre elles. Ensuite lors de la substitution des arbres des groupes nominaux avec l'arbre initial du verbe *aimer*, il se

[15] Le lecteur trouvera des informations sur les grammaires basées sur l'unification dans [MT90].

[16] Les traits *top* sont représentés en exposant et les traits *bottom* en indice.

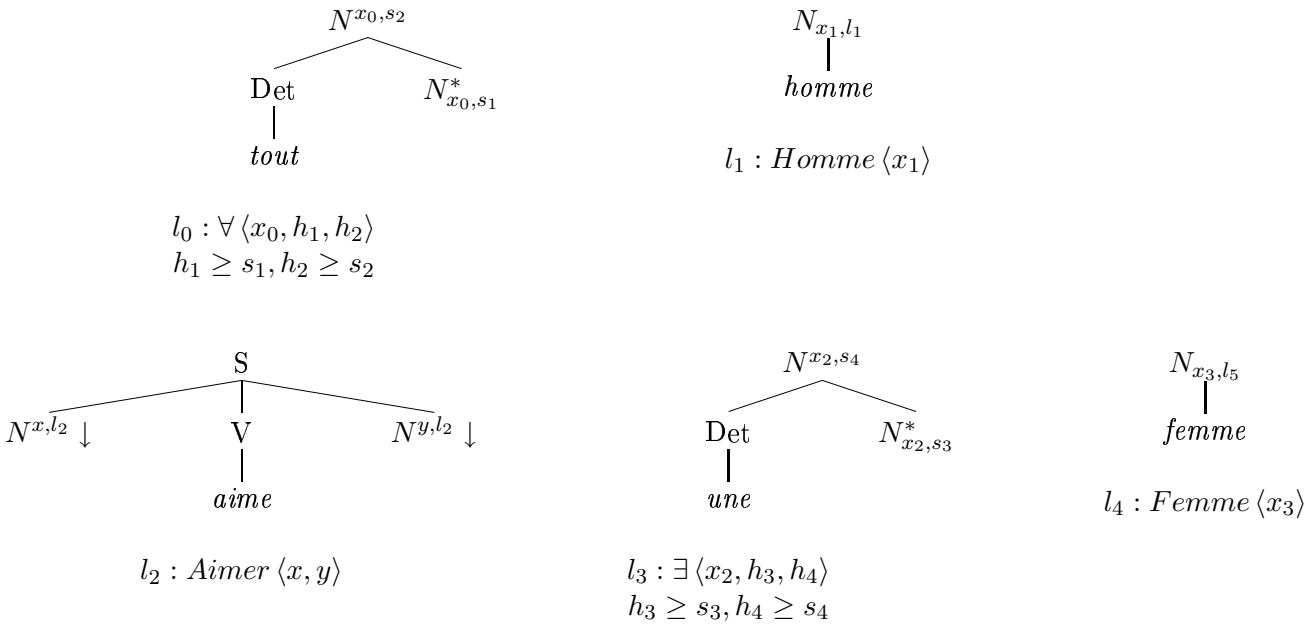
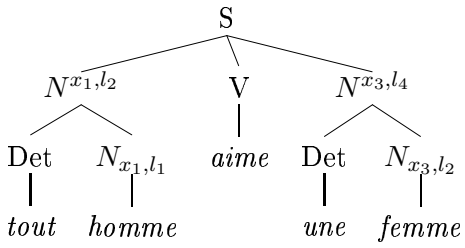


FIG. 1.9 – Exemple de grammaire TAG avec interface syntaxe/sémantique.



$l_0 : \forall(x_1, h_1, h_2)\ ,\ h_1 \geq l_1, h_2 \geq l_2\ ,\ l_1 : Homme(x_1), l_2 : Aimer(x_1, x_2)\ ,$
$l_3 : \exists(x_2, h_3, h_4)\ ,\ h_3 \geq l_4\ ,\ h_4 \geq l_2\ ,\ l_4 : Femme(x_2)$

FIG. 1.10 – Arbre TAG complet avec interface syntaxe/sémantique.

produit encore une unification, afin de mettre à jour les arguments du prédicat *aimer*.
On peut remarquer l'utilisation de la coindexation des traits *index* entre le noeud pied et le noeud racine des arbres auxilaires pour exprimer le lien entre la restriction et la portée des quantificateurs «tout» et «un». La formule L_U ainsi produite correspond bien à celle présentée en 1.3.1.

Cette théorie permet le couplage calcul sémantique / construction syntaxique dans une grammaire TAG représentant des phénomènes linguistiques qui se sont avérés problématiques pour l'approche basée sur l'utilisation de l'arbre de dérivation, comme les quantificateurs (cf ci-dessus), mais aussi les verbes de contrôle et les adjectifs intersectifs (tels que «grand» par exemple). Dans la suite de ce document, nous allons aborder la question de son implémentation dans des outils de génération automatique de grammaires TAG (voir chapitres 3 et 4). Mais avant cela, abordons la notion de méta - grammaire et de compilateur de méta - grammaire.

Chapitre 2

Génération semi-automatique d'une Grammaire d'Arbres Adjoints

2.1 Motivation

En considérant les principes de formation des arbres élémentaires donnés à la section 1.1.3, on remarque que, dans une langue telle que le français, il existe un très grand nombre d'arbres élémentaires (plusieurs milliers). Ce qui conduit inévitablement à plusieurs problèmes :

- l'espace mémoire nécessaire pour stocker la grammaire est important,
- l'écriture d'une grammaire devient difficile (énumération des nombreuses structures syntaxiques existantes, avec une forte redondance d'informations),
- enfin la maintenance de la grammaire se révèle être une tâche très ardue (ajout d'un nouveau trait par exemple),

On a donc réfléchi à un moyen de « factoriser » l'information, en se basant notament sur le constat suivant : de nombreux arbres élémentaires ont des sous-structures communes.
Dans la section qui suit, nous allons présenter les travaux qui ont permis la factorisation de l'information redondante dans les grammaires TAG, ces théories sont à l'origine des techniques exploitées dans les procédés de génération semi-automatique de ces grammaires.

2.2 Théories sous - jacentes

La première remarque que nous pouvons faire est la suivante : plusieurs mots peuvent ancrer des arbres similaires (par exemple l'arbre des verbes transitifs à arguments canoniques est le même que l'ancre soit « aime », « regarde », etc). Comment représenter le lien entre un mot et un schème (i.e. un arbre sans ancre

lexicale) ? La hiérarchisation du lexique adresse cette question[17].

2.2.1 Hiérarchisation et héritage

Ce concept a été introduit dans [VSS92]. On considère deux types d'objets : d'une part des arbres élémentaires et d'autre part des mots (ces mots forment le lexique). Le principe de la hiérarchisation du lexique est le suivant : mettre en relation deux mots ancrant des arbres élémentaires dont certaines sous-structures sont identiques. Dans ce contexte, on définit des classes correspondant à des arbres élémentaires partiellement définis. On utilise en outre l'héritage multiple afin de permettre la réutilisation d'information. Chaque mot se voit associé à une classe. Le lien entre deux éléments du lexique est alors représenté par une relation d'héritage entre classes correspondantes.
Afin d'illustrer cela, un extrait d'une organisation « jouet » du lexique pour les verbes est présenté à la figure 2.1.

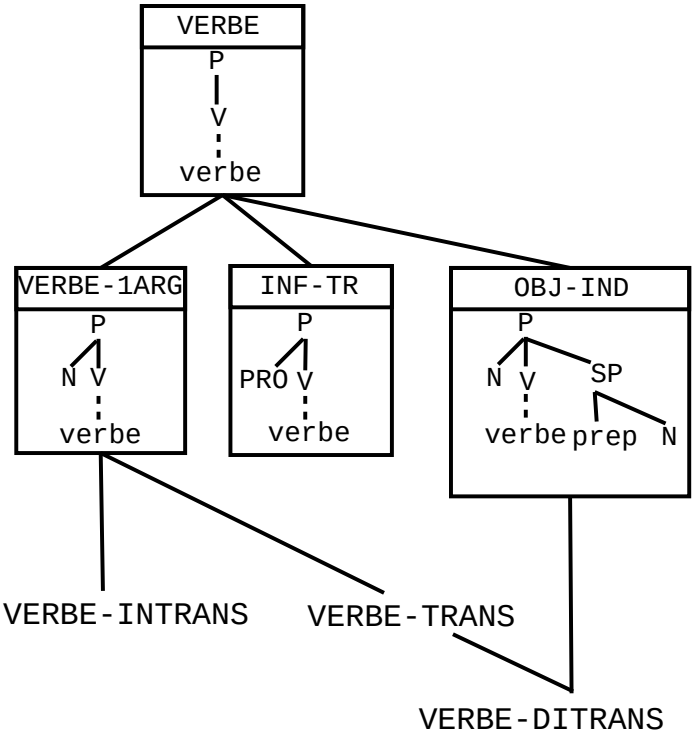


FIG. 2.1 – Exemple d'organisation du lexique : le cas des verbes.

Cet exemple traduit la remarque suivante : tous les arbres élémentaires ancrés par des formes fléchies verbales comportent une branche $P \lhd V \lhd * \ [verbe]$[18]

[17] L'importance de l'organisation du lexique, plus particulièrement dans le cadre de l'analyse syntaxique ou « parsing », a été évoqué dans [Abe90].

[18] Les symboles $\lhd$ et $\lhd *$ représentent respectivement la relation de dominance entre noeuds dans un arbre et la clôture réflexive transitive de cette relation (ou dominance « large »).

(classe VERBE), et parmi les verbes on distingue ceux décrits par des arbres possédant une branche $P \lhd N$ à gauche de la branche $P \lhd V$ (les verbes à un argument sujet - classe VERBE-1ARG), des verbes transitifs à l'infinitif dont les arbres associés ont pour leur part une branche $P \lhd PRO$ à gauche de $P \lhd V$ (classe INFINITIF-TR) et des verbes ayant une branche pour le complément d'objet indirect (classe OBJ-IND). On peut ainsi établir une classification des verbes.

Au sein d'une hiérarchie, une classe est définie par les attributs suivants :

- **Super-classes :** liste des classes situées immédiatement en amont.
- **Noeuds :** liste des noeuds associés à la classe.
- **Description :** description partielle de l'arbre correspondant à la classe. Cette description fait appel à une logique de description d'arbres, comme celle que nous présenterons en 2.2.2.
- **Equations sur les noeuds :** équations représentant des contraintes d'unification sur les traits portés par chaque noeud.
- **Noeud argument :** définition du noeud pour lequel la classe apporte un argument.
- **Précédence linéaire :** définition de l'ordre des noeuds dans le fragment d'arbre, au sens « plus à gauche »[19].
- **Ancre :** Noeud où va être inséré le mot ancrant la structure syntaxique.

La figure 2.2 représente la classe *VERBE-TRANS* de notre hiérarchie jouet, cette classe est associée aux verbes transitifs (e.g. « regarder »).

VERBE-TRANS
Super-classe : VERBE-1ARG
Noeuds : p, v, n
Description : 
Equations : $n.<\text{cat}> = \text{N}$, $n.<\text{accord}> = v.<\text{accord}>$ ...
Noeud argument : n
Précédence : $v < n$
ancre $= v$

FIG. 2.2 – Informations de la classe des verbes transitifs.

Elle hérite directement des informations contenues dans la classe VERBE-1ARG, c'est-à-dire la branche du sujet, et indirectement de celles contenues dans la classe VERBE (branche verbale). Il convient de noter que p, v et n désignent ici des constantes de noeuds et non des catégories syntaxiques.

[19] Cette relation peut être utilisée pour fixer l'ordre de 2 groupes prépositionnels par exemple.

Ainsi nous pouvons partager l'information associée au lexique au moyen de classes et d'héritage. Pour récupérer l'arbre élémentaire associé à un item dans ce contexte, on procède comme suit :

1. on sélectionne la classe associée au composant lexical,
2. on récupère les descriptions partielles d'arbres des classes en amont (héritage de l'information structurelle)[20],
3. on identifie les noeuds de type « ancre» de chacune de ces descriptions,
4. on détermine alors l'arbre correspondant à la conjonction des descriptions, notons que la détermination de cet arbre est liée au langage de description employé (voir 2.2.2).

Enfin, l'emploi de règles syntaxiques et lexicales permet le traitement de transformations telles que le changement de diathèse. Une telle règle associe une entrée lexicale à une autre entrée lexicale. Considérons l'exemple de la figure 2.3 pour la construction de l'arbre élémentaire pour le passif.

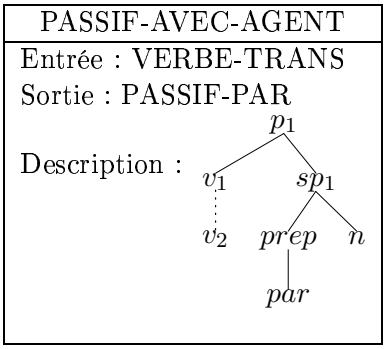


FIG. 2.3 – Règle lexicale pour la construction du passif avec agent.

La règle énoncée permet à partir de la classe VERBE-TRANS, de construire l'arbre élémentaire du passif avec agent (description fournie) et génère la classe PASSIF-PAR. Ces règles constituent un mécanisme non monotone de représentation des structures syntaxiques extérieures à la hiérarchie (représentation des arbres spécifiques au passif ou au phénomène d'extraction par exemple).

2.2.2 Langage de description d'arbres

Dans l'organisation du lexique telle que nous l'avons présentée à la section précédente, on associe à un mot du lexique une description partielle d'arbre. Cette description est donnée dans un langage disposant de la relation sous-spécifiée « $\lhd*$» (dominance large). L'enjeu est alors double : (1) il faut pouvoir calculer l'ensemble des arbres satisfiant une telle description, ce que l'on appelle un modèle, et (2) il faut pouvoir récupérer un arbre de cet ensemble pour lequel toutes les relations sont déterminées (i.e. sans sous-spécification). Pour réaliser cela, [RVS92] décrit un système formel de recherche de modèle à partir

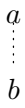
[20]Cela peut nécessiter un renommage des variables de noeuds dans le cas de descriptions partielles employant les mêmes noms de noeuds.

de descriptions exprimées dans un langage disposant des relations entre noeuds suivantes :

- $\lhd$ dominance stricte (appelée aussi parenté),
- $\lhd*$ dominance large,
- $\prec$ précédence (au sens « plus à gauche dans l'arbre », ne s'applique pas entre parents),
- $\approx$ égalité entre noeuds,

et de la convention que tout chemin entre deux noeuds a une longueur finie.

On remarque que la relation de dominance large engendre le problème suivant :



Une description de ce type est vérifiée par une infinité de modèles (ceux ayant un nombre n de noeuds entre a et b avec $n \in [0, \infty[$[21]), ces modèles diffèrent par adjonction d'arbres entre a et b.
Dans ce contexte, nous allons considérer une classe d'équivalence de modèles par rapport à la relation d'adjonction, cette classe sera appelée *quasi-arbre* (termes de [VS92]). La figure 2.4 nous montre un exemple de quasi-arbre. Ce quasi-arbre correspond à l'ensemble des structures syntaxiques incluant l'arbre associé aux verbes transitifs (on prend la conjonction des descriptions partielles associées à la classe VERB-TRANS et à ses super-classes).

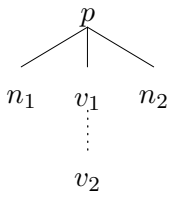


FIG. 2.4 – Exemple de quasi-arbre.

Dans notre langage de description, il est décrit par la formule :

$$p \lhd n_1 \wedge p \lhd v_1 \wedge n_1 \prec v_1 \wedge v_1 \lhd * v_2 \wedge p \lhd n_2 \wedge v_1 \prec n_2$$

Ce quasi-arbre contient deux *quasi-noeuds* v_1 et v_2. Ceux-ci doivent se retrouver dans tout arbre associé à un verbe transitif, et y sont séparés par un chemin de longueur positive ou nulle[22].

[21] On considère les modèles pour lesquels n est fini, cf convention sur la longueur des chemins entre noeuds.

[22] L'intérêt de cette sous-spécification peut également être mis en évidence dans un contexte d'analyse syntaxique où il faut retrouver l'arbre élémentaire du verbe alors que cet arbre peut avoir subi l'adjonction d'un adverbe sur le noeud de catégorie syntaxique V.

La résolution du problème (1) évoqué ci-dessus, à savoir calculer l'ensemble des arbres (ou modèle) satisfiant une description partielle, utilise le langage du 1er ordre suivant[23] :

Définition 4 *Soit K un ensemble fini de constantes, on appelle L_K le langage construit sur K et les symboles suivants :*

- $\lhd, \lhd*, \prec, \approx$ *des prédicats binaires représentant respectivement la dominance stricte, la dominance large, la précédence et l'égalité,*
- $\wedge, \vee, \neg$ *les connecteurs logiques,*
- $[,], (,)$ *symboles de groupement.*

Nous souhaitons construire un modèle consistent avec une description, c'est-à-dire un quasi-arbre exprimé dans notre langage et vérifiant certaines relations.

Définition 5 *Soit un langage L_K, un modèle pour L_K est un 5-uplet $\langle \mathcal{U}, \mathcal{I}, \mathcal{P}, \mathcal{D}, \mathcal{L} \rangle$, où :*

- $\mathcal{U}$ *est un ensemble de noeuds,*
- $\mathcal{I}$ *une fonction de K dans $\mathcal{U}$,*
- $\mathcal{P}, \mathcal{D}, \mathcal{L}$ *des relations binaires interprétant respectivement $\lhd, \lhd*, \prec$.*

Nous avons de plus, pour un certain $\mathcal{R} \in \mathcal{U}$(la racine) et $\forall\ w, x, y, z \in \mathcal{U}$ des contraintes de bonne formation des arbres :

C_1 $\langle \mathcal{R}, x \rangle \in \mathcal{D}$,

C_2 $\langle x, x \rangle \in \mathcal{D}$,

C_3 $\langle x, y \rangle, \langle y, x \rangle \in \mathcal{D} \Rightarrow x = y$,

C_4 $\langle x, y \rangle, \langle y, z \rangle \in \mathcal{D} \Rightarrow \langle x, z \rangle$,

C_5 $\langle x, y \rangle \in \mathcal{P} \Rightarrow \langle x, y \rangle \in \mathcal{D},\ and\ \langle y, x \rangle \notin \mathcal{D}$,

C_6 $\langle x, z \rangle \in \mathcal{P},\ and\ \langle x, y \rangle, \langle y, z \rangle \in \mathcal{D} \Rightarrow x = y\ or\ y = z$,

(...)

Les axiomes C_3 et C_4 correspondent respectivement à la réfléxivité et à la transitivité de la relation de dominance large $\lhd*$.
Dans notre recherche de modèle, nous introduisons certaines notations :

Définition 6 *Soit ϕ un ensemble de formules, T un arbre :*

- $Mod(\phi)$ *est l'ensemble des arbres qui satisfont ϕ,*
- $Th(T)$ *est l'ensemble des formules satisfaites par T,*
- $Cn(\phi) = Th(Mod(\phi))$ *désigne l'ensemble des conséquences de ϕ sur les arbres finis.*

Nous avons vu qu'un quasi-arbre représente une classe d'arbres égaux modulo une adjonction à l'endroit des dominances larges. On remarque de plus qu'en opérant une adjonction sur un arbre T_1, les 2 principes suivants sont respectés :

[23] Les logiques de descriptions d'arbres ne correspondant pas au thème central de notre étude, nous ne donnerons que les définitions et résultats les plus importants de cette théorie, le lecteur trouvera l'ensemble des résultats dans [RVS94].

- l'adjonction n'affecte pas les relations de précédence entre noeuds de T_1,
- deux noeuds distincts de T_1 restent distincts après adjonction.

Nous pouvons alors, à partir de ces propriétés, introduire une relation de pré-ordre partiel $\leq_\phi$ définie par :

Définition 7 *Soient Φ un ensemble de formules, T_1 et T_2 des arbres de $Mod(\phi)$, alors $T_1 \leq_\phi T_2$ ssi $\forall t, u$ tels que t, u satisfassent ϕ :*

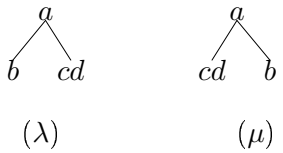
- $T_1 \models t \prec u \Leftrightarrow T_2 \models t \prec u$,
- $T_1 \models \neg(t \lhd * u) \Leftrightarrow T_2 \models \neg(t \lhd * u)$.

Dans ce contexte, si une description Φ n'est pas un quasi-arbre, il existe au moins deux arbres minimaux (au sens de $\leq_\phi$) de $Mod(\Phi)$ incomparables. Considérons par exemple, la description de la figure 2.5, l'ordre entre les noeuds b et c n'est pas spécifié. Cette description se voit associée les 2 arbres minimaux (λ) et (μ). Elle ne correspond donc pas à un quasi-arbre.

Description :

$a \lhd b \wedge a \lhd c \wedge c \lhd * d$

Arbres minimaux :



(λ) (μ)

FIG. 2.5 – Exemple de description ne correspondant pas à un quasi-arbre.

Ce qui signifie que, dans un quasi-arbre, on sait si deux quasi-noeuds sont en relation de précédence ou de dominance (de longueur ≥ 1) l'un par rapport à l'autre. Nous allons donc pouvoir résoudre notre description (i.e. calculer notre modèle) en utilisant un système composé de clauses, branches et configurations :

- les clauses généralisées $[\Phi]$ correspondent à un ensemble de formules interprétées de manière disjonctive (des formules de description),
- les branches $[|\ \Gamma\ |]$ correspondent à un ensemble de clauses généralisées interprétées de manière conjonctive (i.e. des quasi-arbres),
- les configurations Σ correspondent à un ensemble de branches interprétées de manière disjonctive (l'ensemble des quasi-arbres vérifiant la description).

Pour cela, on applique aux branches des règles d'inférence logiques, structurelles, de résolution, et de bonne formation des arbres, comme par exemple la règle logique suivante (règle de la conjonction) :

$$\frac{\Sigma, [|\ \Gamma, [\Phi, \phi \vee \psi]\ |]}{\Sigma, [|\ \Gamma, [\Phi, \phi], [\Phi, \psi]\ |]}$$

cela afin de saturer les configurations (plus de règle applicable, tautologies). Le procédé est supporté par la proposition suivante :

Proposition 1 (Invariance du modèle) *Quelles que soient les dérivations* $\langle \Sigma_0, \Sigma_1, ... \rangle$, $\forall i,\ Mod(\Sigma_i) = Mod(\Sigma_0)$.

Si l'on rencontre une clause vide , la branche correspondante est fermée. Si toutes les branches d'une configuration sont fermées, la description Φ est insatisfiable. Si la description est satisfiable, on aboutit à une configuration ayant au moins une branche valide[24] (i.e. un quasi-arbre), ce quasi-arbre nous donne un modèle dans lequel on sait, pour tout quasi-noeud t, u, si $t \prec u$ ou $\neg(t \prec u)$.

A ce stade, il nous reste à résoudre (2), c'est-à-dire trouver un arbre complètement décrit et satisfiant les relations de notre modèle. Pour cela, on cherche à compléter les branches saturées en traitant les dominances sous-spécifiées. On ajoute la formule : $\forall t, u,\ t \lhd * u$ si cela n'entraîne aucune contradiction, on ajoute ensuite $\forall t, u,\ t \lhd u$ (appliquée aux noeuds en suivant le même ordre) si ce n'est toujours pas contradictoire. On construit alors l'arbre de l'ensemble dans lequel tous les quasi-noeuds dont la relation de dominance large peut être de longueur nulle, sont identifiés (relation $\approx$). Cet arbre est complètement spécifié, c'est l'arbre que nous cherchons. Il est appelé *référent minimal* et le mécanisme de réduction des dominances larges *une lecture circonscriptive* du modèle.

2.3 La notion de méta-grammaire

Dans ce paragraphe, nous allons aborder les principes de base du concept de méta-grammaire, ensuite nous évoquerons deux approches de grammaires électroniques à large couverture qui font référence : la grammaire TAG pour l'anglais développée par le groupe XTAG à l'université de Pennsylvanie, et celle pour le français développée par le laboratoire TALANA de l'université Paris 7.

2.3.1 Principes

Le terme de méta-grammaire (MG) est du à M.H. Candito et est explicité dans [Can99], où l'auteur présente un système de génération semi-automatique de grammaire TAG (voir section 2.3.2 pour une présentation de ce système). Nous avons vu en 2.1 les enjeux de la génération semi-automatique d'une grammaire TAG, à savoir la simplicité d'écriture, la facilité de mise à jour ainsi que l'économie de mémoire. Dans ce but M.H. Candito définit les principes de base d'une méta-grammaire, qui sont les suivants :

- une méta-grammaire intègre une représentation modulaire et hiérarchisée des schèmes[25] d'une grammaire TAG lexicalisée,
- une méta-grammaire permet de générer l'ensemble de ces schèmes,
- une méta-grammaire repose sur des principes linguistiques de hiérarchisation des structures syntaxiques.

Ce qui conduit à la définition suivante :

[24] Si la description est complètement spécifiée, il y aura une unique branche.

[25] Un schème correspond à un arbre élémentaire dans lequel l'ancre lexicale n'est pas instanciée.

Définition 8 (Méta - grammaire) *Une méta - grammaire consiste en un ensemble de règles décrivant les règles de production (i.e. les schèmes) d'une grammaire.*

2.3.2 Approches existantes

Notons que pour pouvoir produire une grammaire (on dit aussi compiler) à partir d'une méta-grammaire, il faut un outil de génération appelé compilateur. Les systèmes présentés ici intègrent ces deux composants[26].

Le groupe XTAG

Le projet XTAG est mené par le département de sciences cognitives de l'université de Pennsylvanie depuis 1987. Bien que consistant à l'origine en une grammaire TAG[27] jouet de l'anglais, il s'est développé pour aboutir dans la première moitié des années 90 à un système intégrant de nombreux outils informatiques (environnement de développement basé sur X-Window, outils de maintenance et analyseur syntaxique) et surtout représentant une grammaire de taille importante.
Un apport important des travaux du groupe XTAG concernant les problèmes liés à l'écriture et la maintenance de la grammaire réside dans la structuration du dictionnaire. En effet dans le projet XTAG, le dictionnaire a 3 niveaux :

- une base de données morphologique,
- une base de données syntaxique,
- et une base de données de schèmes[28] (ou templates dans la littérature anglaise).

L'idée du système XTAG est d'associer des formes fléchies augmentées de traits morphologiques à des lemmes. De plus chaque ensemble formé d'un lemme et de traits morpho-syntaxiques ancre un ensemble de schèmes. Cette organisation multi-niveaux permet de faire abstraction de la morphologie. L'intérêt est visible en analyse syntaxique notament, car l'analyseur peut ainsi construire les arbres TAG dynamiquement lors du parcours de la phrase (seuls les arbres nécessaires sont construits).
Le projet XTAG utilise en outre le concept de famille d'arbres. Une famille d'arbre correspond à l'ensemble des arbres élémentaires associés à un prédicat. On désigne souvent cela par les termes de « cadre de sous-catégorisation», une ancre lexicale sous-catégorise les composants syntaxiques présents dans son arbre élémentaire. Considérons l'exemple suivant :

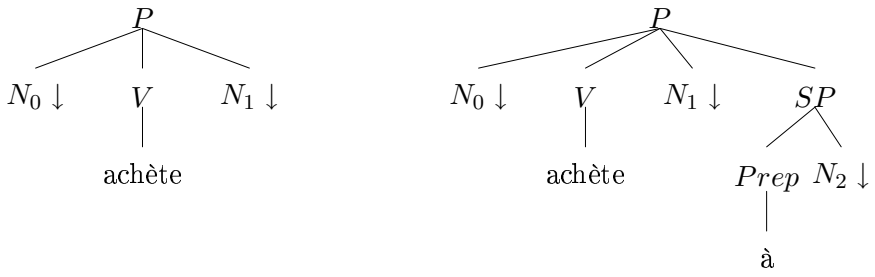
(1) Jean achète une pomme.
(2) Jean achète une pomme à Marie.

[26] Les grammaires TAG développées à L'université de Pennsylvanie et à Paris 7 consistaient à l'origine en une grammaire au sens propre, puisque elles n'intègraient pas de processus de génération semi-automatique de schèmes.

[27] Il faut lire grammaire TAG lexicalisée à structures de traits.

[28] Dans un schème, le mot ancre désigne le noeud sous lequel l'item lexical va prendre place, alors que, dans un arbre élémentaire, il désigne le noeud de l'item lexical.

Dans (1) le verbe *acheter* sous-catégorise un sujet et un complément d'objet direct (une pomme) correspondant au noeud de catégorie N situé à droite du verbe dans l'arbre élémentaire. Dans (2) ce même verbe *acheter* sous-catégorise un sujet et deux compléments d'objets comme on peut le voir figure 2.6.



(Famille N_0VN_1) (Famille $N_0VN_1\text{à}N_2$)

FIG. 2.6 – Deux sous-catégorisations du verbe acheter.

La base de données des schèmes est en réalité subdivisée en familles de schèmes, et chaque lemme ancre une certaine famille.

D'un point de vue technique, le système XTAG comporte des modules écrits en Prolog (module de résolution des descriptions d'arbres), interfacés notamment avec l'environnement X-Window via des scripts Perl[29].

Afin d'avoir une meilleure idée de la couverture de la grammaire du projet XTAG, la figure 2.7 nous donne quelques informations chiffrées.

Composant	**Description**
Base de données morpholgique	environ 317 000 formes fléchies
Base de données syntaxique	environ 30 000 entrées
Base de données de schèmes	1094 schèmes regroupés en 52 familles plus 218 schèmes individuels.

FIG. 2.7 – Informations sur le projet XTAG.

L'université Paris 7

La méta-grammaire développée par M.H. Candito dans la seconde moitié des années 90 repose sur une critique des autres approches sur le point suivant : l'organisation du lexique en classes avec héritage multiple doit suivre des principes linguistiques.

[29]Pour des informations complémentaires, le lecteur pourra consulter le site du projet XTAG à l'adresse *http ://www.cis.upenn.edu/~xtag*.

C'est ainsi qu'a été fait le choix de représenter le lexique au moyen d'une hiérarchie d'héritage monotone, ce qui d'une part permet d'exprimer les relations entre structures syntaxiques dans une hiérarchie déclarative plus facilement interprétable, et d'autre part interdit l'emploi de règles lexicales telles qu'énoncées dans [VSS92]. Les exceptions au schéma classique de formation des arbres élémentaires (cas du passif par exemple) sont dorénavant traités par le biais d'une organisation tridimensionnelle des descriptions syntaxiques :

- **dimension 1 : sous-catégorisation initiale**, cette dimension contient une hiérarchie représentant les différentes sous-catégorisations canoniques (par exemple N_0VN_1 pour un schème associé à un verbe transitif),
- **dimension 2 : redistributions des fonctions syntaxiques**, les classes de cette dimension représentent les redistributions possibles de fontion syntaxique des arguments du cadre de sous-catégorisation initiale (par exemple la classe *objet* $\rightarrow$ *sujet* (utilisée pour exprimer le passif) assigne la fonction sujet à l'argument objet de la sous-catégorisation initiale),
- **dimension 3 : réalisation des fonctions syntaxiques**, cette 3^{e} dimension exprime les différentes réalisations possibles pour une fonction syntaxique (par exemple sujet clivé « c'est Jean qui dort », objet interrogé « que mange Jean ? »).

Précisons quelques points importants à ce stade :

- les informations contenues dans les différentes classes de cette hiérarchie consistent entre autre en des descriptions partielles exprimées dans un langage semblable à celui présenté en 2.2.2. Dans ces descriptions partielles, les noeuds sont désignés par des constantes de manière à pouvoir référer au même noeud dans 2 classes différentes,
- plusieurs types d'héritage sont employés dans cette hiérarchie : on dispose d'un héritage par défaut, d'un héritage multiple, et d'un héritage prioritaire,
- des structures de traits typées sont associées à certaines constantes de noeud.

A partir de ce schéma d'organisation, on calcule l'ensemble des schèmes du langage par un procédé de croisement particulier :
On réalise le produit cartésien des classes terminales de la dimension 1 avec les classes terminales de la dimension 2, on obtient ainsi un ensemble de sous-catégorisations finales (i.e. des couples de classes). Pour chaque couple, on crée la classe héritée (qui contient notamment la conjonction des descriptions des classes mères). Enfin pour chaque fonction syntaxique *fs* apparaissant dans cette sous-catégorisation finale, on la croise avec chacune des classes de dimension 3 réalisant *fs*. On dispose alors des descriptions des différents schèmes de la grammaire, descriptions que l'on peut résoudre comme vu en 2.2.2.
Au cours de ce procédé de génération des schèmes, on élimine les classes croisées qui ne vérifient pas certaines contraintes de compatibilité (de telles contraintes permettent par exemple d'exprimer qu'un verbe intransitif ne supporte pas la voix passive).
L'exemple de la figure 2.8 nous montre comment la MG génère un schème[30]

[30] Le symbole $\diamondsuit$ désigne l'ancre du schème.

(l'ancrage de ce schème se fait en utilisant un dictionnaire similaire à celui du groupe XTAG).

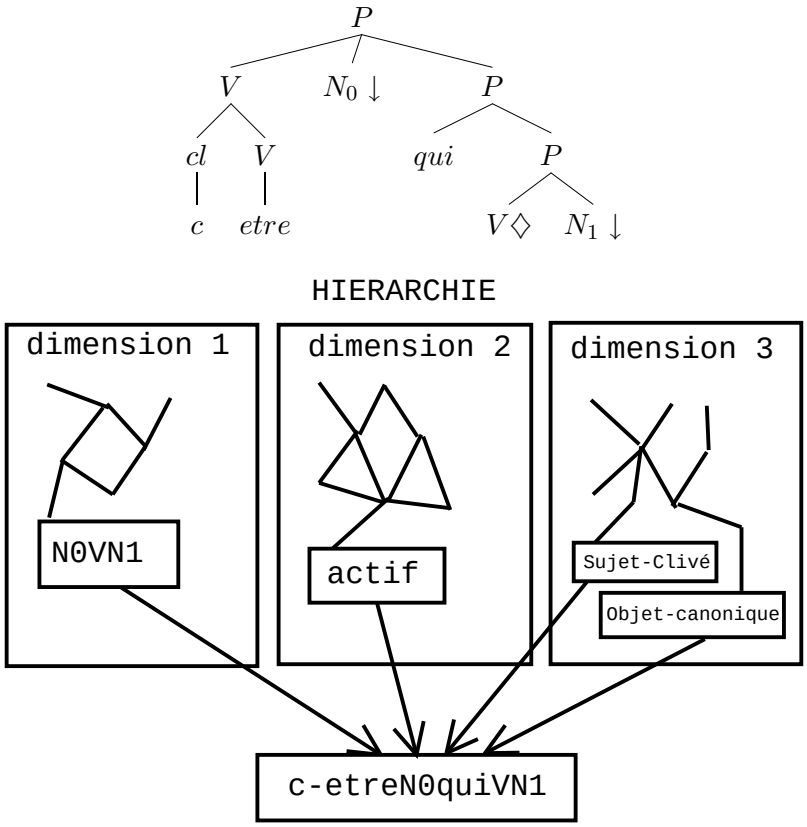


FIG. 2.8 – Génération du schème $c - etreN_0quiVN_1$.

Précisons enfin que la méta-grammaire du français comporte 54 familles d'arbres pour les verbes, et elle permet la génération de 5240 schèmes (sans le causatif)[31]. La grammaire résultant de la compilation contient de plus 40 schèmes non-générés automatiquement, ces schèmes correspondent aux déterminants, conjonctions de coordination, clitiques, verbes déficients, auxilaires, et certains noms. Techniquement, le compilateur de cette méta-grammaire est écrit en langage Common LISP.

[31] Source : [ACK99].

Chapitre 3

L'exemple du compilateur de méta-grammaire MGC

3.1 Présentation

Le compilateur de méta-grammaires (MGC) présenté dans ce chapitre a été développé au sein de l'équipe Langue et Dialogue par Bertrand Gaiffe, Benoît Crabbé et Azim Roussanaly[32]. Il est issu d'une part d'une réflexion sur la structuration du lexique syntaxique et d'autre part des critiques du compilateur de M.H. Candito sur les points suivants :

- manque de généricité : l'algorithme de génération des schèmes est spécifique à l'organisation tridimensionelle de représentation des structures syntaxiques,
- implémentation rigide : elle ne permet pas la modification des formats d'entrée / sortie à un faible coût.

Dans un premier temps nous allons définir la structure d'une méta-grammaire avec MGC, puis nous nous intéresserons plus particulièrement à l'intégration d'une représentation sémantique dans ce cadre, enfin nous déroulerons un exemple de génération automatique de grammaire avec sémantique par MGC.

3.1.1 Hiérarchisation de l'information

Lors de la programmation du compilateur MGC, les objectifs suivants ont été posés :

1. pouvoir décrire une organisation des descriptions syntaxiques ayant un nombre quelconque de dimensions,
2. garantir un méchanisme monotone de génération de schèmes.

En outre le compilateur associe à chaque schème généré une structure de traits globale, l'intérêt étant de pouvoir décrire précisément ce schème afin de faciliter l'ancrage de l'item lexical. Cette structure de traits globale doit se retrouver dans notre hiérarchie, cela passe par la présence dans une classe d'une structure de traits spécifique appelée *hypertag*.

[32]MGC est disponible à l'adresse *http ://www.loria.fr/~gaiffe*.

L'écriture de la méta-grammaire suppose la définition d'une ou plusieurs hiérarchies de classes. Au sein de chaque hiérarchie, les classes sont en relation d'héritage multiple. Une classe dans ce contexte est définie par les attributs suivants :

- un nom,
- un ensemble de super-classes (classes dont elle hérite),
- une structure de traits *hypertag*,
- un ensemble de besoins (symboles atomiques),
- un ensemble de ressources (symboles atomiques),
- un ensemble éventuellement vide de quasi-noeuds désignés par des constantes[33],
- un ensemble éventuellement vide de formules de descriptions[34] utilisant les quasi-noeuds de la classe et ceux hérités,
- un ensemble d'équations exprimant des contraintes de coindexation entre traits associés aux quasi-noeuds.

Techniquement, la logique employée dans les descriptions partielles désigne les noeuds dans les arbres par des constantes. Lorsqu'on conjoint deux formules utilisant une même constante, cette constante désigne le même noeud. L'emploi de contantes de noeuds permet donc la désignation du même noeud dans deux classes en relation d'héritage. En outre, chacune de ces constantes de noeuds se voit associer un type (noeud de substitution, pied, ancre, etc), une catégorie syntaxique (N, V, etc), une structure de traits *top* et une structure de traits *bottom*.
Dans une hiérarchie, lorsqu'une classe hérite de deux classes c_1 et c_2, elle reçoit pour formule de description la conjonction des formules de description associées à c_1 et c_2, ses besoins (respectivement ressources) correspondent à l'union des besoins de c_1 et c_2 privé de l'union de leurs ressources (respectivement l'union des ressources de c_1 et c_2 privé de l'union de leurs besoins), enfin son hypertag est issu de l'unification des hypertags de c_1 et c_2.
Voyons à présent comment, à partir de ces hiérarchies, le compilateur génère l'ensemble des schèmes.

3.1.2 Procédé de combinaison des classes

Dans notre schéma d'héritage, on aboutit à des classes terminales (c'est-à-dire situées à l'extrémité d'une hiérarchie) pouvant contenir des descriptions de schèmes incomplètes. Afin de les compléter, il faut récupérer l'information manquante dans d'autres classes terminales. Pour cela, nous allons effectuer des croisements entre classes terminales, par croisement nous entendons un héritage multiple tel que défini en 3.1.1. Ce croisement génèrera des classes que nous appelerons finales.

[33] Un utilise le terme de quasi-noeud pour indiquer que ce noeud n'est pas forcément désigné par une unique constante.

[34] Il s'agit ici des formules dans une logique propositionnelle interprétée sur les arbres finis, c'est-à-dire dans lesquels deux noeuds sont séparés par un chemin de longueur finie.

Le problème est de contraindre ces croisements afin de garantir (1) la **consistence** des schèmes décrits (la classe d'un verbe intransitif ne doit pas être croisée avec la classe du passif par exemple) ainsi que (2) leur **complétude** (le croisement générant un schème verbal doit comporter une classe décrivant la fonction sujet[35]).

La consistence (point (1)) est réalisée par l'emploi des structures de traits hypertag. Lors du croisement de deux classes finales, on unifie ces structures. On peut donc bloquer un croisement en utilisant par exemple un système de polarités (on étiquette la classe des verbes intransitifs avec les hypertags [TRANSITIF = -] et [PASSIF = -], et celle des classes supportant une construction à la voix passive avec [TRANSITIF = +] et [PASSIF = +][36]).

Concernant la complétude (point (2)), il nous faut un procédé « forçant » deux classes à se croiser. On utilise pour cela une notion de *Besoins / Ressources*. Considérons une classe contenant la description d'un verbe transitif, elle *nécessite* la réalisation de la fonction grammaticale sujet et de la fonction objet. Elle va donc se croiser avec d'une part une classe *fournissant* une réalisation du sujet et d'autre part une classe *fournissant* une réalisation de l'objet (i.e. des classes fournissant entre autres la description des branches associées au sujet et à l'objet).

Le croisement consiste à calculer les parties « équilibrées » de l'ensemble des classes terminales C, c'est-à-dire les ensembles de classes C_i dans lesquels chaque besoin est annulé par la présence d'une ressource correspondante dans une des classes de C_i, **et** chaque besoin n'apparaît qu'une fois dans C_i, **et** chaque ressource n'apparaît qu'une fois dans C_i ; puis à générer la classe finale héritant des classes de C_i.
La figure 3.1 nous montre un exemple simple de croisement de classes en fonction des Besoins / Ressources.

A l'issue du méchanisme, on dispose, pour chaque classe finale, d'une description à partir de laquelle on dérive le schème correspondant par le procédé de recherche du référent minimal présenté en 2.2.2. Le schème ainsi obtenu est en outre décoré d'une structure de trait globale. Cette structure va permettre de désigner précisément le schème dans le dictionnaire lors de la définition de l'ancrage pour une entrée lexicale. La classe finale de l'exemple de la figure 3.1 ainsi que le schème et l'hypertag associés sont représentés sur la figure 3.2.

Précisons enfin que dans MGC, le dictionnaire regroupe le lexique syntaxique, le lexique morphologique et les schèmes, l'analyseur syntaxique réalise l'ancrage de manière dynamique via une équation d'ancrage (voir section 3.3).

[35] Chaque fonction grammaticale doit être réalisée une et une seule fois au sein d'un croisement.

[36] On utilise ici deux hypertags car certains verbes transitifs ne supportent pas la voix passive, exemples : « comporter », « peser ».

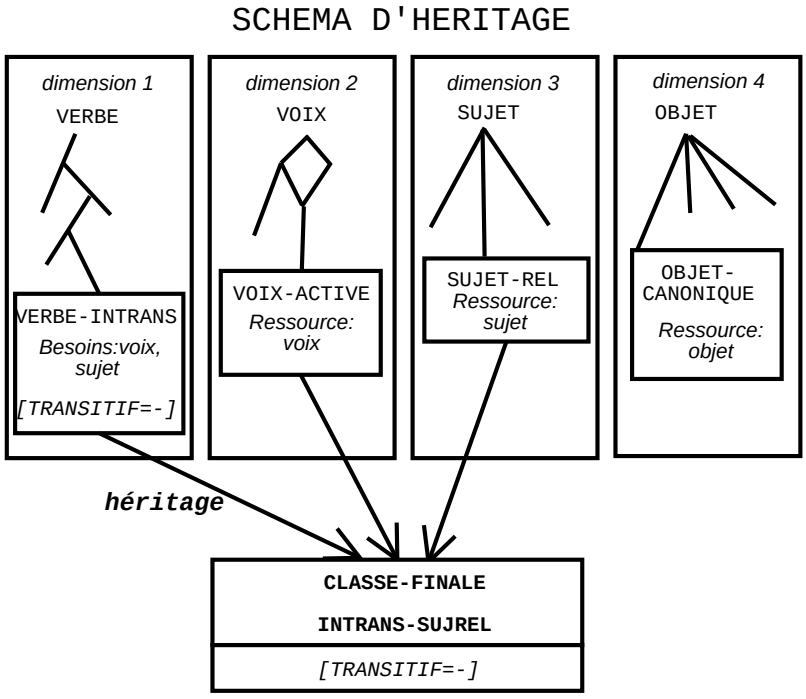


FIG. 3.1 – Exemple de croisement dans MGC.

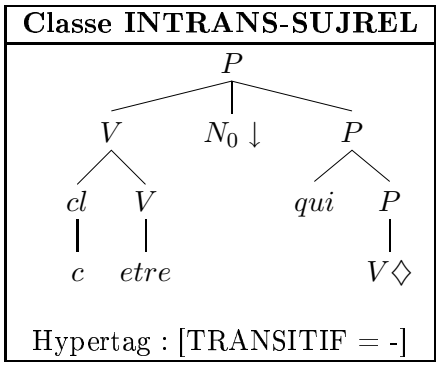


FIG. 3.2 – Résultat d'un croisement dans MGC.

3.2 Intégration du calcul sémantique

Dans cette section, nous allons aborder l'interfaçage entre syntaxe et sémantique dans MGC. Plus précisément, nous allons voir comment intégrer l'information sémantique dans la méta-grammaire fournie au MGC afin de générer une grammaire TAG associant aux arbres élémentaires une représentation sémantique telle que celle présentée en 1.3.

3.2.1 « Noeud sémantique »

Pour intégrer les informations sémantiques dans notre méta-grammaire, c'est-à-dire les associer à notre schéma de classes, nous devons résoudre deux difficultés :

(1) trouver où placer la formule de sémantique plate associée à un arbre élémentaire,

(2) trouver comment partager les variables de cette formule avec les traits sémantiques associés à certains noeuds des arbres élémentaires[37].

Concernant (1), deux solutions sont envisageables, à savoir, associer une structure de traits supplémentaire à chaque classe (du type hypertag) et qui décorera le schème créé avec une formule sémantique (issue de la composition des formules des différentes classes héritées et croisées), ou placer les informations de la formule sémantique dans des traits spécifiques appelés *sem* et placés par exemple dans la structure *top* d'un certain noeud.
La 1ère possibilité énoncée impose de tenir compte de cette structure de trait sémantique « globale » lors de la recherche des arbres élémentaires combinés, pendant la phase d'analyse syntaxique. Cela suppose une modification relativement importante des analyseurs existants. Dans l'implémentation que nous avons réalisé au LORIA, nous avons donc privilégié la 2e solution car elle ne nécessite aucune adaptation de l'analyseur syntaxique. Ce dernier restituera un arbre dérivé contenant toute l'information sémantique.
Comment choisir le noeud qui va contenir les informations de la formule sémantique ? Il faut porter attention au fait que les structures de traits *top* et *bottom* sont soumises à unification lors de la combinaison des arbres élémentaires, on ne doit pas permettre qu'une combinaison échoue par la présence de traits sémantiques incompatibles (i.e. ayant des valeurs différentes pour les attributs *sem*). Le premier réflexe est de choisir le noeud ancre pour placer l'information sémantique (et pas le noeud de l'item lexical car nous générons des schèmes et non des arbres élémentaires, et les traits sémantiques vont devoir être partagés avec d'autres noeuds). Cependant le problème d'unification des structures de traits se pose également dans ce cas, prenons l'exemple d'un adverbe comme « beaucoup » dont l'arbre élémentaire va s'adjoindre au noeud ancre de l'arbre[38] du verbe « manger » pour générer une expression telle que « manger beaucoup ». Afin de parer à ce problème, nous avons fait le choix d'utiliser un noeud de

[37] Il faut porter attention au fait que la définition des différents noeuds à information sémantique n'intervient pas forcément dans la même classe au sein de nos hiérarchies.

[38] En réalité au noeud ancre du schème, le terme ancre désignant habituellement, dans un arbre élémentaire, le noeud de l'item lexical.

« catégorie »[39] *Sem*. Ce noeud *Sem* est situé sous le noeud qui servait d'ancre, et devient noeud ancre. C'est sur lui que vont venir se placer les items lexicaux.

3.2.2 Structures de traits non atomiques

A présent que nous avons décidé d'inclure la formule sémantique dans les schèmes, la première idée concernant (2) est d'utiliser d'une part les structures de traits *top* et *bottom* associées aux quasi-noeuds de nos classes pour contenir les variables de la formule sémantique, et d'autre part le procédé de coindexation pour partager l'information contenue par ces traits avec le noeud *Sem* du schème.
Il convient cependant de remarquer certains points :

- En règle générale, les grammaires TAG utilisent des structures de traits atomiques afin d'engendrer des langages disposant de certaines propriétés (cf section 1.2).

- Les outils utilisant les grammaires TAG (analyseurs syntaxiques et compilateurs) sont implémentés pour des traits atomiques, mais peuvent être relativement facilement adaptés au traitement de traits non-atomiques.

- Dans certains cas l'information à associer à un schème consiste en plusieurs prédicats, c'est le cas des quantificateurs notamment. Comme nous l'avons vu à la section 1.3, en sémantique plate, à un quantificateur est associée une relation de quantification ainsi que deux relations de portée. Il est donc préférable de disposer de traits complexes afin de distinguer les arguments des différents prédicats. La figure 3.3 nous présente le schème associé à un quantificateur. Le noeud *Sem* contient la structure de traits incluant l'information sémantique. Le trait *top.head* contient le prédicat (« forall » ou « exists ») avec ses arguments et les traits *top.tail.head* et *top.tail.tail* les relations de portée $\geq$ (désignées par « ScopeOver ») concernant respectivement le 1^{er} et le 2^{nd} argument du prédicat principal.

- Enfin, si nous choisissons d'« étaler » les arguments sémantiques dans des structures de traits atomiques, la récupération de la formule de sémantique plate lors de l'analyse demandera un traitement supplémentaire.

Dans ce contexte, nous avons un choix à faire, à savoir respecter la contrainte théorique qui veut que les grammaires TAG aient des traits atomiques, ce qui peut impliquer des problèmes d'implémentation (récupération de la formule logique) ou bien passer outre cette contrainte et adapter l'analyseur et le compilateur aux traits complexes. La formule de sémantique plate sera alors restituée lors de l'analyse, au sein du noeud *Sem*, et avec l'ensemble de ses variables à jour. C'est cette dernière option que nous avons sélectionnée dans notre approche.

[39] A proprement parler, on ne peut pas employer le terme de catégorie ici.

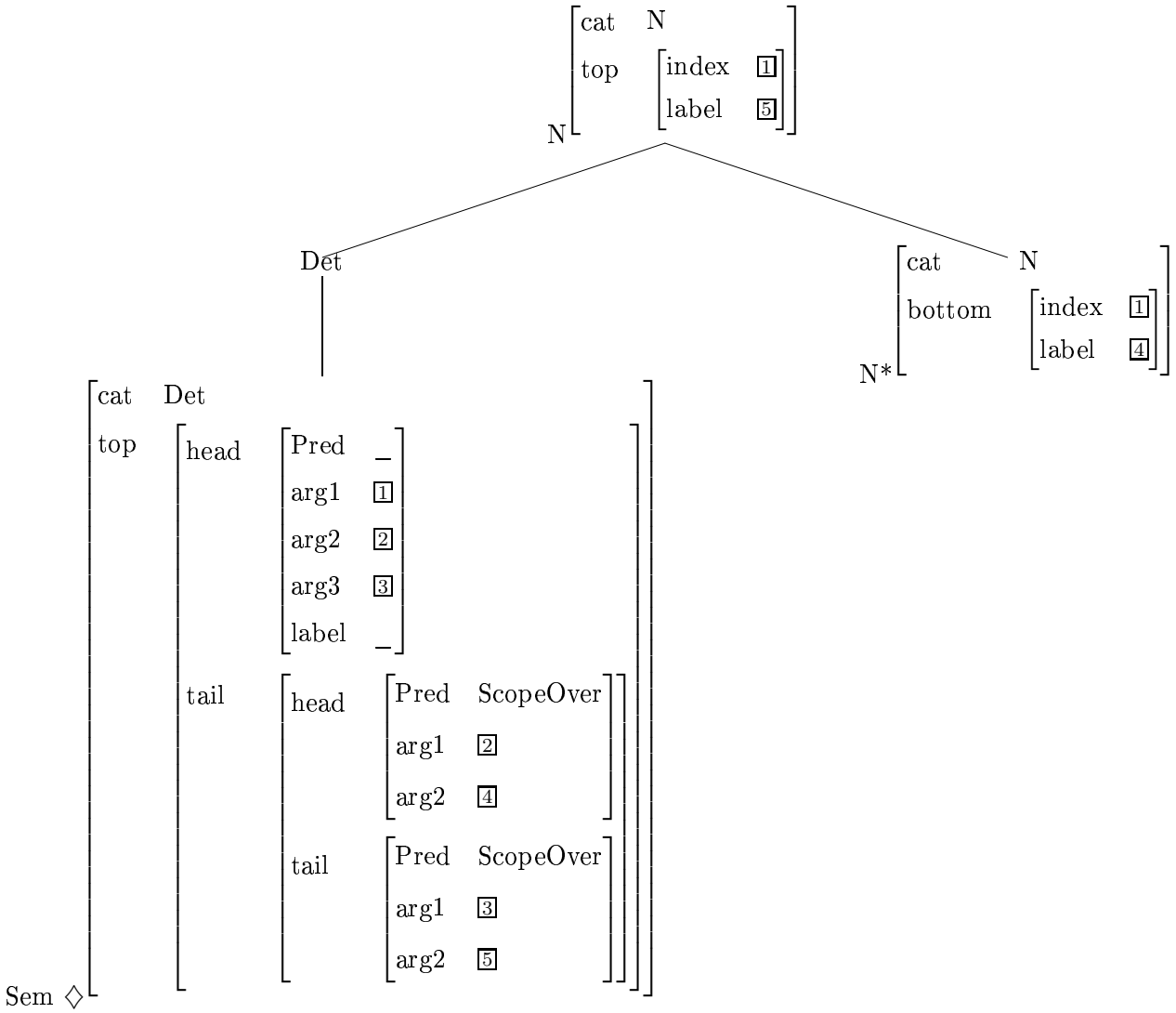


FIG. 3.3 – Schème associé à un quantificateur.

3.3 Application : écriture d'une petite méta-grammaire à portée sémantique pour MGC

A présent, nous allons voir concrètement comment générer une petite grammaire TAG intégrant une représentation sémantique avec MGC.
Tout d'abord présentons brièvement l'outil utilisé. MGC a été développé en langage Java et contient bien entendu un compilateur de méta-grammaire mais également un éditeur de méta-grammaire. En parallèle au développement de MGC, a été codé un visualisateur d'arbres, ainsi qu'un analyseur syntaxique. Les méta-grammaires sont stockées dans un fichier au format XML, et les schèmes produits dans un fichier au format TagML2[40].
Le procédé de génération d'une grammaire TAG à informations sémantiques est le suivant :

1. définition d'une ou plusieurs hiérarchies de classes permettant une factorisation de l'information liée au schèmes (rappel : ces classes contiennent les descriptions partielles d'arbres),
2. affectation des Besoins / Ressources et de l'hypertag pour guider le croisement des classes,
3. coindexation des traits sémantiques sur les noeuds concernés,
4. compilation de la méta-grammaire pour produire un ensemble de schèmes stocké dans un fichier,
5. écriture du lexique syntaxique et du lexique morphologique (le prédicat est remonté sur le noeud sémantique du schème via une équation d'ancrage contenue dans le lexique syntaxique),
6. regroupement des lexiques et des schèmes au sein du dictionnaire.

A l'issue du procédé, nous disposons d'une méta-grammaire utilisable par l'analyseur syntaxique pour calculer l'arbre TAG dérivé correspondant à une phrase.

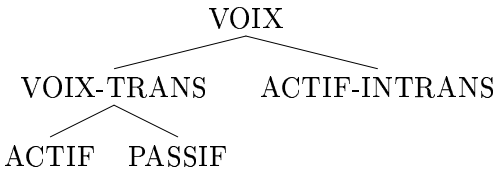
Première étape : écriture des hiérarchies.[41] Nous allons écrire une méta-grammaire permettant de représenter les verbes intransitifs et transitifs, les voix active et passive avec agent[42], les quantificateurs, ainsi que les noms. Concernant ces deux derniers, il n'y a pas à proprement parler de hiérarchie, puisqu'on définit directement une classe contenant la description complète du schème qui leur est associé. Par contre, concernant les verbes, nous pouvons factoriser l'information entre structures syntaxiques au moyen, par exemple, de la hiérarchie présentée figure 3.4.

[40] Le format TagML2 est un format dédié aux ressources linguistiques, basé sur XML et développé au sein de l'équipe Langue et Dialogue.

[41] L'appel à l'éditeur de MGC se fait via la commande « runme.sh -edit ».

[42] Le traitement du passif sans agent nécessite l'introduction d'une noeud « vide », nous n'évoquerons pas ce cas dans un souci de clarté.

La classe VERBE contient la description de la branche $P \lhd V \lhd Sem$, la classe VERBE-1ARG définit de plus la présence d'un noeud N[43] sous le noeud P sans spécifier l'ordre de la branche $P \lhd N$ par rapport à celle décrite dans VERBE. De même on définit la classe des verbes transitifs comme comportant un deuxième noeud N sous le noeud P sans spécifier d'ordre. On remarque la présence de la classe VERBE-INTRANS héritant de VERBE-1ARG, cette classe n'apporte pas d'information complémentaire par rapport à VERBE-1ARG, elle a été définie car seules les classes terminales sont croisées. Rappelons que N représente dans nos propos la catégorie syntaxique du noeud, mais celui-ci est en réalité désigné par une constante (nommée Arg1Node - pour l'argument 1 du verbe - et Arg2Node dans notre exemple). A présent il faut une réalisation des arguments du verbe (c'est-à-dire indiquer sur quels noeuds vont se trouver le sujet et l'objet), cela se définit lorsque l'on décrit la voix, c'est donc dans la hiérarchie ci-dessous que l'on va spécifier les relations de précédence manquantes :



Dans cette hiérarchie, les schèmes associés au passif réutilisent la description associée à la voix active en y ajoutant les branches pour l'auxiliaire *être* et la préposition *par*. En outre les classes de cette hiérarchie définissent des noeuds nommés par exemple SujNode et ObjNode, qui vont être décrit par des relations telles que $[SujNode \approx Arg1Node\ ,\ ObjNode \approx Arg2Node]$ pour la voix active et $[SujNode \approx Arg2Node\ ,\ ObjNode \approx Arg1Node]$ pour la voix passive (voir figure 3.5).

Deuxième étape : définition des Besoins / Ressources et des hypertags. Afin de s'assurer que les bons croisement vont être effectués, il convient d'utiliser les *Besoins / Ressources* de la manière suivante : tous les verbes vont avoir pour besoin le symbole « voix », besoin qui sera fourni par la classe VOIX via la ressource « voix » (héritée par ses filles). De plus on spécifiera que la classe d'un verbe transitif a un hypertag [TRANSITIF = +], que la classe du passif également, et celle des verbes intransitifs [TRANSITIF = -], afin de s'assurer qu'on ne croisera pas de verbe intransitif avec une classe du passif. La figure 3.5 illustre cela.

Troisième étape : définition des coindexations entre traits sémantiques. A présent que nous avons défini une organisation des descriptions permettant une factorisation de l'information syntaxique, nous pouvons placer nos traits sémantiques aux différents noeuds concernés. Pour cela, nous allons utiliser les équations entre traits associées à chaque classe. Prenons l'exemple d'un verbe transitif tel que "aimer", le schème qui lui est associé aura l'allure de la figure 3.6.

[43] Nous nous intéresserons uniquement aux verbes dont les arguments sont nominaux.

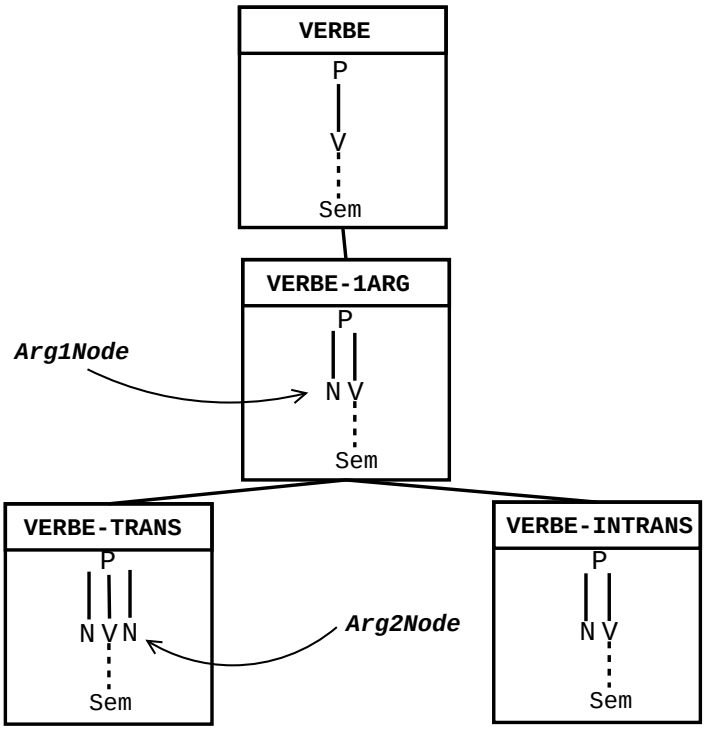


FIG. 3.4 – Exemple de hiérarchie d'héritage dans MGC.

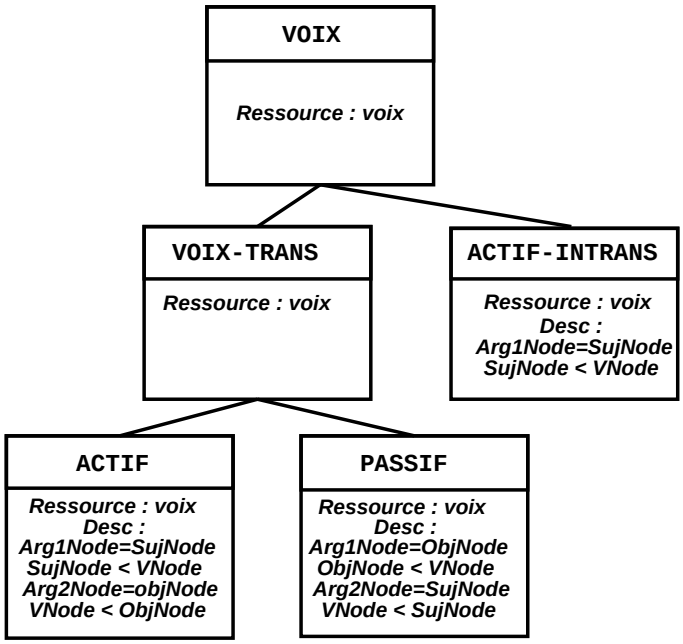


FIG. 3.5 – Définition des Besoins/Ressources dans MGC.

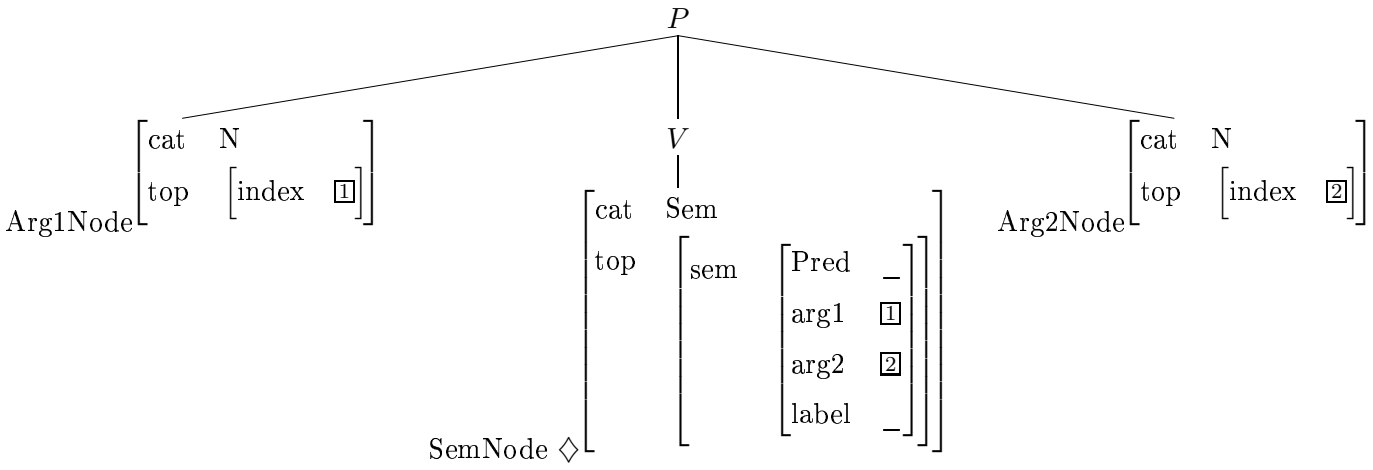


FIG. 3.6 – Schème associé à un verbe transitif.

Le noeud *Sem* va contenir la formule L_U de la représentation sémantique. Dans cette formule, les arguments du prédicat sont coindexés avec les traits *top.index* des noeuds de catégorie N. Pour représenter cela dans notre méta-grammaire, nous allons utiliser les équations dites de chemin suivantes :

$$Arg1Node.top.index = SemNode.top.sem.arg1$$
$$Arg2Node.top.index = SemNode.top.sem.arg2$$

Quatrième étape : compilation de la méta-grammaire. Une fois notre petite méta-grammaire écrite, elle est sauvegardée dans un fichier au format XML, nous pouvons alors lancer sa compilation via MGC par la ligne de commande « runme.sh -tagml2 MaMetaGrammaire.mg ». Des informations sur les classes lues et les croisements effectués sont imprimés sur la sortie standard, enfin le nombre d'arbres générés est affiché. Les arbres créés sont sauvegardés au format TagML2 dans le fichier « ./results/Trees.tagml ».

Cinquième étape : définition des lexiques syntaxiques et morphologiques. Le lexique syntaxique va associer un lemme à une famille de schèmes, famille qui sera désignée par un certain hypertag ([TRANSITIF = +] pour la famille des verbes transitifs par exemple). Ce lexique syntaxique est contenu dans un fichier au format tagml2. Cependant afin de faciliter l'écriture de ce lexique, un analyseur spécifique (dit de lexicalisation) a été écrit. Cet analyseur prend en entrée un fichier texte contenant les informations liées aux lemmes et génère le fichier tagml2 correspondant. Voici l'entrée lexicale associée au lemme "aimer" (on remarque l'utilisation de l'équation d'ancrage pour remonter l'information sémantique *prédicat* sur le schème) :

```
# Begin
# Lexeme aimer
# Acception 1
# Families [TRANSITIF = +]
# Anchors
```

```
SemNode = aimer Sem
# Equations
SemNode [top = [sem = [Pred = aimer]]]
# End
```

et sa transcription au format tagml2 :

```
<lexicalization>
  <family>
  <tree><fs>
  <f name="TRANSITIF">
  <sym value="+"/>
  </f></fs></tree></family>
<anchor noderef="SemNode">
<lemmaref name="aimer" cat="Sem"/></anchor>
<equation type="top" noderef="SemNode">
  <fs><f name="sem">
  <fs><f name="Pred">
  <sym value="aimer"/></f></fs></f></fs></equation>
<equation type="bot" noderef="SemNode">
<fs/></equation></lexicalization>
```

Nous devons ensuite écrire le lexique morphologique. Ce lexique associe les formes fléchies aux lemmes définis dans le lexique syntaxique. Il est écrit dans un fichier au format tagml2. Concernant son écriture, il n'existe pas à l'heure actuelle d'outil équivalent à l'analyseur utilisé précédemment, il faut donc écrire les entrées directement en tagml2. Voici un exemple d'entrées du lexique morphologique :

```
<morph lex="aime">
  <lemmaref cat="Sem" name="aimer"/>
</morph>
<morph lex="aiment">
  <lemmaref cat="Sem" name="aimer"/>
</morph>
```

Sixième étape : regroupement des lexiques et des schèmes dans le dictionnaire. Le dictionnaire correspond à un fichier au format tagml2, dans lequel sont copiés les entrées des lexiques (syntaxique et morphologique), ainsi que les schèmes générés par MGC. Nous pouvons alors appeler l'analyseur syntaxique, puis via le menu *Fichier / Ouvrir*, charger le dictionnaire, et recupérer les arbres dérivés associés aux phrases couvertes par notre grammaire.
Le lecteur trouvera des illustrations du procédé de génération automatique de grammaire TAG à portée sémantique via MGC et de celui d'analyse syntaxique avec une telle grammaire en annexe.

Dans ce chapitre, nous avons découvert une suite d'outils complète, permettant d'une part la génération automatique d'une grammaire TAG intégrant une

représentation sémantique, et d'autre part l'analyse de phrases en langue naturelle. L'intérêt de MGC réside entre autres dans sa souplesse d'utilisation, il permet d'écrire et de compiler des méta-grammaires aux formats divers. Il fait face à une combinatoire importante lors des croisements de classes, ainsi que lors de la construction des schèmes, et malgré cela se révèle d'une certaine puissance. Cependant, il présente un inconvénient relativement important, comme nous allons le voir au chapitre 4, qui réside dans l'utilisation de constantes de noeuds. Celles-ci rendent difficile l'écriture et l'évolution d'une méta-grammaire à large couverture à partir d'un certain seuil.

Chapitre 4

L'exemple du résolveur de descriptions colorées

4.1 Présentation

Nous allons à présent nous intéresser à un nouveau type de compilateur de méta-grammaire : le résolveur de descriptions colorées. Ce compilateur est en cours de développement dans le cadre de la thèse de Benoît Crabbé en partenariat avec le Docteur Denys Duchier de l'équipe Calligramme.
Lors de travaux basés sur la génération de grammaire TAG à large couverture pour le français au moyen des compilateurs de M.H. Candito et MGC, un problème est apparu concernant l'écriture et la maintenance de la méta-grammaire : il s'agit de l'utilisation de constantes de noeuds dans les descriptions. Nous avons vu en 2.2 et en 3.1 que les noeuds des arbres étaient désignés par des constantes afin de permettre leur réutilisation dans différentes classes de la hiérarchie. L'emploi de ces constantes est relativement lourd lorsque l'on veut écrire une méta-grammaire de taille importante. Pour parer à cela, Benoît Crabbé a proposé un procédé de description des structures syntaxiques dans lequel les noeuds sont anonymes (voir 4.1.1). Ce procédé utilise une logique de description faisant appel à des variables de noeuds combinées par un code de couleurs. Des noeuds colorés ont déjà été utilisés pour l'analyse syntaxique (cf [DT99]), de plus un logiciel de résolution de description développé à l'université de Sarrebrück est disponible (cf [DG99]). Nous nous en sommes inspirés.
Dans cette section de présentation, nous allons aborder successivement comment est représentée et factorisée l'information dans une méta-grammaire destinée à être compilée par le résolveur, puis comment ce dernier génère les arbres de notre grammaire, et enfin comment a été implémenté le processus de compilation.

4.1.1 Description d'arbres

Des autres approches de représentation des structures syntaxiques d'une grammaire TAG (i.e. les arbres), nous conservons l'idée de hiérarchie d'héritage. Cependant, l'utilisation de descriptions dont les noeuds sont anonymes nous permet d'effectuer une factorisation plus forte en hiérarchisant non plus

l'information liée aux schèmes mais celle liée à des fragments d'arbres[44]. Pour être plus clair, dans une méta-grammaire telle que celles vues jusqu'à présent, nous cherchions à factoriser un schème en sous-spécifiant certaines relations entre noeuds (voir figure 4.1(a)), alors que dans l'approche du résolveur, nous allons "découper" les arbres élémentaires et placer les fragments d'arbres dans une hiérarchie d'héritage unique (voir figure 4.1(b)).

Dans notre schéma d'héritage, chaque classe contient les informations suivantes[45] :

- un nom,
- une description de fragment d'arbre exprimée dans un langage de description disposant des relations entre variables de noeuds suivantes : $\lhd$, $\lhd*$, $<$ et $\ll$ représentant respectivement la dominance stricte, la dominance large, la précédence, et l'adjacence (ou précédence immédiate) entre frères.

On remarque que les fragments d'arbres de notre hiérarchie correspondent en fait à des réalisations syntaxiques de prédicats et d'arguments.
Définissons clairement l'héritage dans le cas présent. Il s'agit d'un héritage unique, chaque classe a au plus une classe mère, si une classe C_1 hérite d'une classe C, alors :

$$desc(C_1, C) = desc(C_1) \wedge desc(C)$$

$$nom(C_1, C) = concat(nom(C_1), nom(C))$$

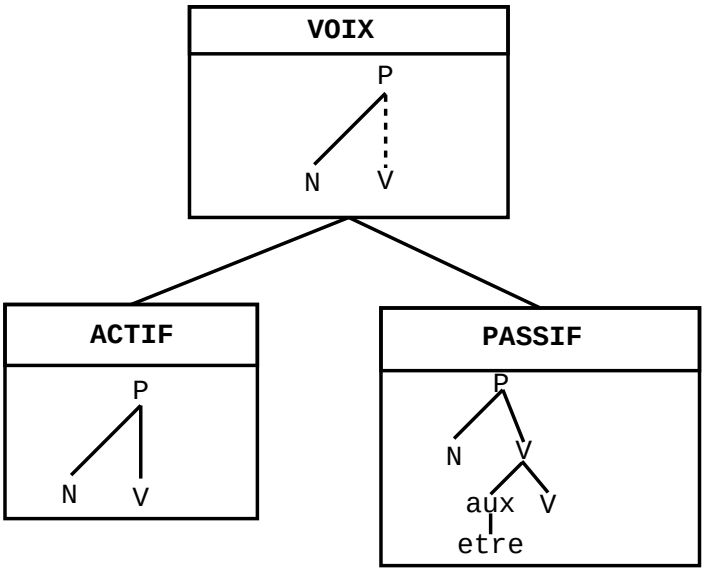
Le nom de la classe fille reprend le nom de C auquel on concatène une chaîne afin de le "spécialiser" (exemple : classe mère SUJ pour sujet, et classe fille $SUJ - CAN$ pour sujet canonique).
A ce stade, on dispose d'une hiérarchie de forme arborescente. Afin de reconstituer les arbres élémentaires que doit générer notre méta-grammaire, nous devons encore combiner les classes feuilles entre elles. Il y a là deux difficultés : (1) combiner les classes "compatibles", et (2) calculer les arbres associés à ces combinaisons. On remarque que dans l'approche MGC par exemple, la combinaison de classes compatibles se voit associée des descriptions référant aux mêmes noeuds (nommés), ce qui permet un calcul "classique"[46] de l'arbre associé, ce n'est pas le cas du résolveur.

Concernant la combinaison des classes feuilles (point (1)), nous allons définir des combinaisons explicites à partir d'un langage de description puissant intégrant la conjonction ($\wedge$) et la disjonction ($|$). Un exemple de combinaison de classes feuilles est celui permettant la génération des arbres ancrés par les verbes transitifs à la voix active. Dans les phrases faisant intervenir un tel verbe, chaque actant grammatical (sujet, objet, etc) peut se réaliser de différentes manières telles que (pour le sujet) : sujet canonique, sujet relativé, sujet questionné, etc, ce qui se traduit par la disjonction SUJET = SUJ-CAN | SUJ-REL | SUJ-QU | ... La combinaison VERBE-TRANS-ACTIF correspond à la

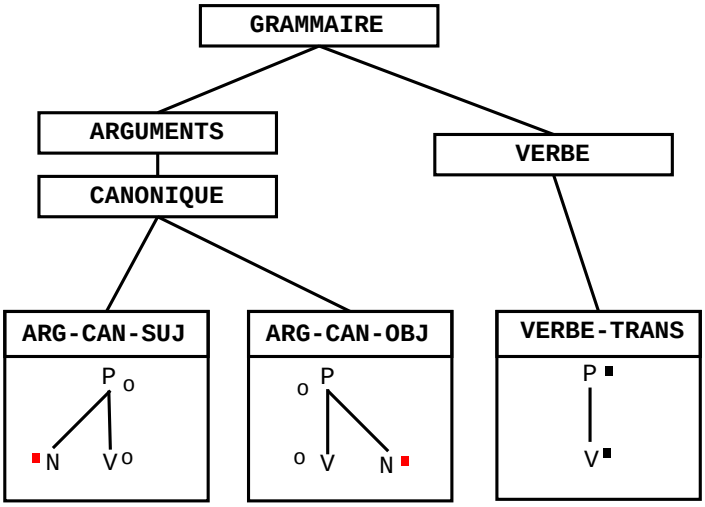
[44] Nous employons ici le terme *arbre* volontairement, car le résolveur produit des arbres élémentaires et non des schèmes comme nous le verrons à la section 4.2.

[45] Nous ne décrivons pas ici les informations sémantiques associées à une classe, cela sera traité en 4.3.

[46] Méthode de calcul du référent minimal.



(a) Schéma d'héritage dans une méta-grammaire « classique ».



(b) Schéma d'héritage dans une méta-grammaire à noeuds colorés.

FIG. 4.1 – Factorisation d'information dans les méta-grammaires TAG.

conjonction : VERBE-TRANS-ACTIF = SUJET $\wedge$ ACTIF $\wedge$ OBJET (on remarque que l'on a une conjonction de disjonctions, i.e. un produit cartésien du type {SUJET}×{ACTIF}×{OBJET}).
Regardons concrètement comment nous écrivons cet exemple dans notre implémentation de méta-grammaire à noeuds colorés. La notation employée pour définir une classe est la suivante :

$$\text{class X} = \text{E}$$

$$\textit{où } E ::= \begin{cases} E|E & (\textit{disjonction de classes}) \\ E\&E & (\textit{conjonction de classes}) \\ < syn > \{...\} & (\textit{spécification d une description syntaxique}) \end{cases}$$

La définition d'un fragment d'arbre se fait en utilisant la notation suivante :

```
<syn> {
      node(couleur,type)[A1=V1,..,An=Vn] {
      ...node{...}
         node{...}
      ,,,node{...}
      }
}
```

Le premier « `node` » réfère à la racine du fragment, les autres « `node` » aux sous-noeuds. Par défaut un noeud domine strictement un sous-noeud, les « ... » servent à spécifier une dominance large. De même, par défaut chaque sous-noeud précède immédiatement le suivant dans la liste, sauf si l'on utilise « ,,, », auquel cas la précédence est large. Il faut faire attention à l'emploi de la dominance large dans nos descriptions car la précédence immédiate s'applique uniquement à des frères (i.e. des noeuds de même étage).
La classe SUJ-CAN de notre exemple se défini comme suit[47] :

```
class SUJ-CAN = <syn>{
      node(white)[cat = p]{
            node(red,subst)[cat=n, top = [num = ?N, pers=?M]]
            node(white)[cat = v, top=[num=?N, pers=?M]]
      }
}
```

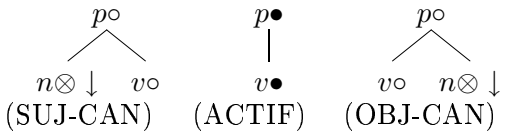
On remarque la présence des traits *num* et *pers* coindexés entre noeuds V et N afin de marquer l'accord entre le sujet et le verbe. Les classes ACTIF et OBJ-CAN sont données respectivement par :

```
class ACTIF = <syn>{
      node(black)[cat=p]{
          node(black)[cat=v]
      }
```

[47] Le ! indique une constante et ? une variable.

```
}
class OBJ-CAN = <syn>{
      node(white)[cat = p]{
            node(white)[cat = v]
         ,,,node(red,subst)[cat=n]
      }
}
```

Ces trois classes représentent respectivement les 3 fragments d'arbres suivants[48] :



La conjonction de ces trois classes forme un élément du produit cartésien VERBE-TRANS-ACTIF. Il en résulte une description correspondant à la conjonction des descriptions de ces fragments. La résolution de cette description (point (2)) se fera au moyen d'un procédé basé sur un langage de noeuds colorés.

4.1.2 Résolution des descriptions : langage de noeuds colorés

Comme nous l'avons vu plus-haut, la méta-grammaire présentée ici repose sur une hiérarchie de descriptions de fragments d'arbres dans lesquels les noeuds sont anonymes. Comment savoir où se placent ces fragments les uns par rapport aux autres dans les arbres que nous souhaitons générer ? Afin de répondre à cette question, nous allons employer un langage reposant sur des noeuds colorés. Dans nos descriptions, nous allons associer à chaque noeud une couleur choisie parmi {rouge, noir, blanc}. Dans ce cadre, la résolution d'une description consiste à identifier les noeuds qui doivent l'être. Les règles d'identification sont données à la figure 4.2[49].

$$
\begin{array}{ccccc}
\circ & + & \circ & \rightarrow & \circ \\
\circ & + & \bullet & \rightarrow & \bullet \\
\circ & + & \otimes & \rightarrow & \perp \\
\bullet & + & \bullet & \rightarrow & \perp \\
\bullet & + & \otimes & \rightarrow & \perp \\
\otimes & + & \otimes & \rightarrow & \perp
\end{array}
$$

FIG. 4.2 – Règles d'identification des noeuds.

Les variables de noeud blanches dénotent des noeuds qui vont être identifiés avec des noeuds dénotés par des variables noires, les variables de noeud rouges dénotent des noeuds non indentifiables. L'utilisation linguistique est que la couleur noir permet de désigner les noeuds qui apportent de l'information alors que

[48] Nous omettons les structures de traits pour ne pas alourdir le schéma.
[49] Le symbole $\otimes$ représente la couleur rouge et $\perp$ indique une combinaison interdite.

la couleur blanche sert à marquer les noeuds constituant un "squelette" d'arborescence. La couleur rouge quant à elle sert à définir un noeud qui ne doit être identifié avec aucun autre (i.e. un noeud ne nécessitant aucun apport d'information).
Pour récapituler le procédé, nous construisons une hiérarchie de classes contenant la description d'un fragment d'arbre, puis, à l'aide de descriptions conjonctives et disjonctives, nous combinons les classes feuilles de cette hiérarchie. Ce qui nous donne une description correspondant à une conjonction de descriptions de classes feuilles, que nous résolvons en identifiant certains noeuds de manière à obtenir un arbre *valide*. Un arbre sera considéré comme valide s'il ne contient plus que des noeuds noirs ou rouges et satisfait les relations de descriptions.
La figure 4.3 nous présente la construction de l'arbre associé aux verbes transitifs à l'actif avec arguments canoniques.

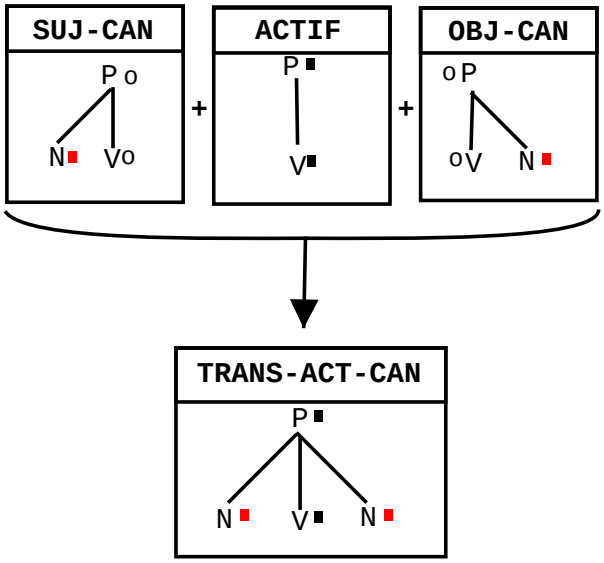


FIG. 4.3 – Combinaison de classes dans une hiérarchie à noeuds colorés.

Il convient de remarquer deux points délicats dans ce procédé de génération d'arbres :

1. le calcul des disjonctions de classes (par exemple : VERBE-ACTIF = SUJET + ACTIF + OBJET avec SUJET = SUJ-CAN | SUJ-CLI | ...) engendre une combinatoire importante (complexité exponentielle),
2. le calcul des arbres résultants de chaque combinaison également.

Afin de borner le processus de compilation, on ajoute des contraintes de bonne formation[50] des arbres générés (via l'introduction de règles générales conditionnant la résolution d'une description, comme l'ordre des noeuds de catégorie *clitique* ou l'unicité de l'extraction par exemple[51]), ainsi que des contraintes de combinaisons de classes (voir la notion d'alternance définie en 4.2.2).

[50] Au sens TAG, cf 1.1.3.
[51] Ces règles sont encore en cours d'implémentation dans le résolveur, en conséquence nous

4.1.3 Technique d'implémentation : programmation par contraintes sur des ensembles

Concernant la résolution des descriptions à noeuds colorés, nous nous sommes basés d'une part sur un résolveur existant pour les logiques de description de base (voir [DG99]) et implémenté par programmation par contraintes sur des ensembles (ce type de programmation, présenté dans [Duc99], s'avère particulièrement bien adapté à la résolution de descriptions à base de dominance), et d'autre part sur un analyseur utilisant des polarités ($\approx$ couleurs, voir [DT99]). Dans ce contexte, la résolution d'une description ϕ consiste en la traduction des relations de ϕ en contraintes sur des ensembles d'entiers. Ces contraintes sont de deux types : (1) des contraintes de bonne formation des arbres, et (2) des contraintes spécifiques aux couleurs.

Contraintes de bonne formation des arbres

Concernant (1), nous allons associer à chaque noeud de notre description un entier i, un noeud N^i sera alors référencé au moyen d'un 5-uplet $(Eq, Up, Down, Left, Right)$ dont les éléments sont des ensembles d'entiers représentant respectivement l'ensemble des noeuds égaux, situés au-dessus, au-dessous, à gauche et à droite de N^i [52] (voir figure 4.4).

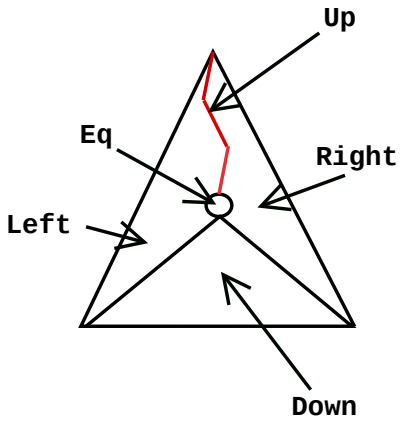


FIG. 4.4 – Référencement d'un noeud par partitionnement d'un arbre.

On remarque que la position d'un noeud est complètement définie par les cinq ensembles $Eq, Up, Down, Left, etRight$. Cependant, à des fins d'optimisation, nous introduisons de plus les ensembles *Parent* et *Children*, ainsi que les ensembles *EqUp*, *EqDown*, *et Side* définis par les contraintes suivantes[53] :

n'en parlerons pas dans la suite de ce rapport.

[52] $Eq, Up, Down, Left, Right$ forment une partition de l'ensemble des noeuds, de plus on a la contrainte $i \in N^i_{Eq}$.

[53] Le symbole $\uplus$ représente l'union disjointe.

$$\begin{array}{rcl} N^i_{EqUp} & = & N^i_{Eq} \uplus N^i_{Up} \\ N^i_{EqDown} & = & N^i_{Eq} \uplus N^i_{Down} \\ N^i_{Side} & = & N^i_{Left} \uplus N^i_{Right} \end{array}$$

Nous pouvons alors exprimer la relation entre deux noeuds de notre description au moyen de contraintes sur ces ensembles :

$$\begin{array}{rcl} N^i \lhd N^j & \equiv & [N^i_{Eq} = N^j_{Parent} \wedge N^i_{EqUp} \subseteq N^j_{Up} \wedge N^i_{Children} \supseteq N^j_{Eq} \wedge \\ & & N^i_{Down} \supseteq N^j_{EqDown} \wedge N^i_{Left} \subseteq N^j_{Left} \wedge N^i_{Right} \subseteq N^j_{Right}] \\ N^i \lhd * N^j & \equiv & [N^i_{EqDown} \supseteq N^j_{EqDown} \wedge N^i_{EqUp} \subseteq N^j_{EqUp} \wedge N^i_{Side} \subseteq N^j_{Side}] \\ N^i < N^j & \equiv & [N^i_{EqDown} \subseteq N^j_{Left} \wedge N^i_{Right} \supseteq N^j_{EqDown} \wedge N^i_{Right} \supset N^j_{Right} \wedge \\ & & N^i_{Left} \subset N^j_{Left}] \\ N^i \ll N^j & \equiv & [N^i_{Right} = N^j_{EqDown} \uplus N^j_{Right} \wedge N^i_{Left} \uplus N^i_{EqDown} = N^j_{Left}] \end{array}$$

En outre, nous allons nous assurer que le modèle décrit par ces relations correspond bien à un arbre. Pour que ce soit le cas, il faut que pour toute paire de noeuds $(N^i,\ N^j)$, $N^i,\ N^j$ soient dans l'une des relations exclusives suivantes : $N^i = N^j$, $N^i \lhd + N^j$, $N^i + \rhd N^j$, $N^i < N^j$ ou bien $N^i > N^j$. Nous exprimons cela au moyen d'une variable C^{ij} et de clauses traduisant la relation entre N^i et N^j :

$$\begin{array}{rcl} N^i = N^j & \wedge & C^{ij} = 1 \ \vee \ C^{ij} \neq 1 \ \wedge \ N^i \neq N^j \\ N^i \lhd + N^j & \wedge & C^{ij} = 2 \ \vee \ C^{ij} \neq 2 \ \wedge \ N^i \not\lhd + \ N^j \\ N^j \lhd + N^i & \wedge & C^{ij} = 3 \ \vee \ C^{ij} \neq 3 \ \wedge \ N^j \not\lhd + \ N^i \\ N^i < N^j & \wedge & C^{ij} = 4 \ \vee \ C^{ij} \neq 4 \ \wedge \ N^i \not< N^j \\ N^j < N^i & \wedge & C^{ij} = 5 \ \vee \ C^{ij} \neq 5 \ \wedge \ N^j \not< N^i \end{array}$$

La variable C^{ij} est contrainte dans l'intervalle [1..5], lors de la recherche de la relation entre deux noeuds dans notre arbre, nous pouvons éliminer plusieurs possibilités infructueuses en associant une des alternatives à cette variable C^{ij}. Par exemple si deux noeuds N^i et N^j sont égaux, la variable C^{ij} vaut 1 et les autres relations entre ces deux noeuds deviennent inconsistentes (C^{ij} étant déterminée, on ne les testera plus). Si Ni et N^j sont distincts, alors nous fixons que $C^{ij} \neq 1$ (i.e. $C^{ij} \in [2..5]$), ce qui restreint l'espace de recherche.

Contraintes spécifiques aux couleurs

Ces contraintes sont exprimées par notre description. Elles concernent les règles d'identification des noeuds présentées à la figure 4.2. Il s'agit de traduire les relations entre noeuds colorés (i.e. les combinaisons de couleurs autorisées) en contraintes sur des ensembles. Par exemple, nous pouvons spécifier qu'un noeud N dont la couleur est rouge ne vas être identifié avec aucun autre : $Card(N_{Eq}) = 1$.
Pour exprimer ces containtes, nous ajoutons à la définition d'un noeud N^i les valeurs suivantes[54] : $theRB$ (pour « the Red-Black ») qui est l'entier représentant le noeud de couleur noire ou rouge avec lequel N^i est identifié (si N^i est noir

[54] En réalité, nous utilisons également les champs *IsRoot* et *UpCard* afin d'optimiser la recherche de la racine.

ou rouge alors $N^i_{theRB} = i$), ainsi que $ChildrenRB$ représentant l'ensemble des noeuds fils de N^i qui sont de couleur noire ou rouge.
Avant de décrire les contraintes spécifiques qui ont été implémentées, nous devons expliciter la notion de contrainte "sélective". La notation abstraite de cette contrainte est $S = < S_1 ... S_n > [S_i]$. Elle a pour but de contraindre la sélection d'un ou plusieurs ensembles parmi une liste (ou plus exactement un vecteur) d'ensembles suivant un critère particulier. Dans la formule énoncée, la valeur de l'ensemble S contraint la valeur de l'ensemble S_i parmi les indices des ensembles S_1 à S_n consistents avec S. Il existe également une contrainte d'union sélective $S = \cup < S_1 ... S_n > [S_i]$, exprimant la contrainte que S est l'union des ensembles S_j pour $j \in S_i$.
On note $\mathcal{T}$, $\mathcal{N}$, $\mathcal{R}$ et $\mathcal{B}$ respectivement l'ensemble de tous les noeuds, des noeuds noirs, rouges et blancs ; i et j sont des entiers. On définit alors les contraintes spécifiques suivantes :

$$\forall i \in (\mathcal{N} \cup \mathcal{R}) \quad N^i_{theRB} = i \tag{1}$$

$$\forall i \in \mathcal{R} \quad Card(N^i_{Eq}) = 1 \tag{2}$$

$$\biguplus_{i \in \mathcal{N}} N^i_{Eq} = \bigcup_{j \in (\mathcal{B} \cup \mathcal{N})} N^j_{Eq} \tag{3}$$

$$Racine \subseteq (\mathcal{N} \cup \mathcal{R}) \tag{4}$$

$$\forall i \in \mathcal{T} \quad N^i_{Children} \subseteq N^i_{Down} \tag{5}$$

$$\forall i \in \mathcal{T} \quad N^i_{ChildrenRB} = N^i_{Children} \cap (\mathcal{N} \cup \mathcal{R}) \tag{6}$$

$$\forall i \in \mathcal{T} \quad N^i_{Children} = \cup < \bigcup_{j \in (\mathcal{N} \cup \mathcal{R})} N^j_{Eq} > [N^i_{ChildrenRB}] \tag{7}$$

$$\forall i \in \mathcal{T} \quad N^i_{Down} = \cup < \bigcup_{j \in (\mathcal{N} \cup \mathcal{R})} N^j_{EqDown} > [N^i_{ChildrenRB}] \tag{8}$$

$$\forall i \in \mathcal{B} \quad N^i_{Eq} = < \bigcup_{j \in (\mathcal{N} \cup \mathcal{R})} N^j_{Eq} > [N^i_{theRB}] \tag{9}$$

$$\forall i \in \mathcal{B} \quad N^i_{Parent} = < \bigcup_{j \in (\mathcal{N} \cup \mathcal{R})} N^j_{Parent} > [N^i_{theRB}] \tag{10}$$

$$\forall i \in \mathcal{B} \quad N^i_{Down} = < \bigcup_{j \in (\mathcal{N} \cup \mathcal{R})} N^j_{Down} > [N^i_{theRB}] \tag{11}$$

$$\forall i \in \mathcal{B} \quad N^i_{Up} = < \bigcup_{j \in (\mathcal{N} \cup \mathcal{R})} N^j_{Up} > [N^i_{theRB}] \tag{12}$$

Nous ne revenons pas sur les contraintes (1), (2), (4), (5) et (6) qui semblent assez explicites. La contrainte (3) indique que l'union disjointe des ensembles Eq des noeuds noirs doit être égale à l'union des ensembles Eq des noeuds noirs et des noeuds blancs, puisqu'à l'issu de la résolution, il ne doit pas rester de noeud blanc non identifié avec un noeud noir et que deux noeuds noirs ne peuvent pas s'identifier. Les contraintes (7) et (8) indiquent que le champ $Children$ d'un noeud N^i (respectivement $Down$) correspond à l'union sélective des champs Eq (respectivement $EqDown$) des noeuds noirs ou rouges contenant des éléments de $N^i_{ChildrenRB}$. Enfin les contraintes (9) à (12) correspondent à des contraintes

sélectives sur les ensembles Eq, $Parent$, $Down$ et Up des noeuds blancs suivant la valeur du noeud noir avec lequel ils sont identifiés (N^i_{theRB}).

4.2 Intégration du calcul sémantique

L'idée que nous avons pour l'interfaçage syntaxe / sémantique dans une méta-grammaire pour le résolveur est d'associer une formule logique à un arbre, et de projeter les arguments de cette formule sur les noeuds appropriés des arbres, en suivant à nouveau l'approche de [GK03]. Cependant nous devons faire face au problème suivant : comment projeter ces informations sur les "bons" noeuds puisque les noeuds de notre description sont anonymes ? Pour résoudre cela, nous allons utiliser le concept de *fonction grammaticale* (e.g. sujet, objet, etc).

4.2.1 Fonction grammaticale

Nous avons vu précédemment que le découpage des arbres en fragments au sein de notre hiérarchie correspondait à un découpage suivant les réalisations syntaxiques de prédicats et d'arguments. Nous pouvons donc identifier les fragments portant l'information sémantique. Plus exactement nous pouvons spécifier quels noeuds représentent le sujet, l'objet, etc. Pour réaliser l'interfaçage syntaxe / sémantique, nous allons utiliser des fonctions grammaticales permettant le repérage des noeuds. Lors de la définition de fragments d'arbres, nous allons affecter aux noeuds porteurs de l'information sémantique un nom symbolique correspondant à sa fonction grammaticale. Un exemple de hiérarchie utilisant les fonctions grammaticales est donné à la figure 4.5.

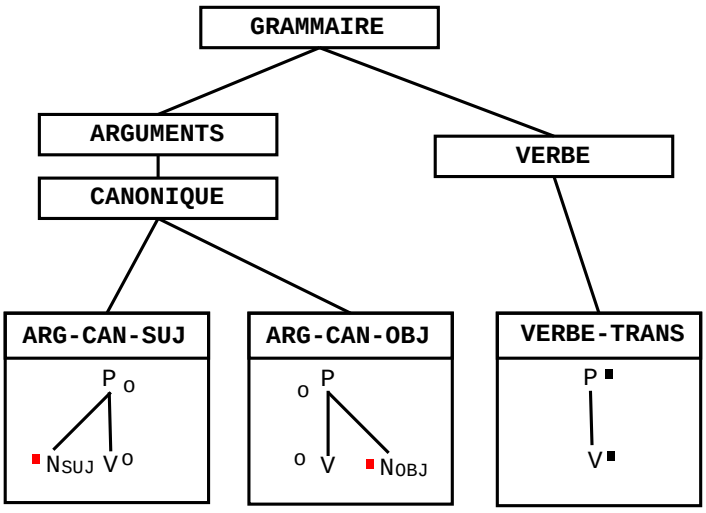


FIG. 4.5 – Affectation de fonctions grammaticales à certains noeuds.

Une fois que l'on connaît les noeuds à coindexer avec les arguments séman-

tiques, il reste à associer la formule de sémantique plate à un arbre. Pour cela, nous allons utiliser un procédé différent de celui implémenté dans MGC, nous n'utiliserons pas de noeud *Sem* mais un couple *(Arbre, Représentation sémantique)*. L'association structure prédicative / réalisations syntaxiques (étape de coindexation) se fera au moyen d'alternances.

4.2.2 Alternance

Une alternance consiste en l'association d'un léxème[55] à un ensemble de descriptions (i.e. un ensemble d'arbres). On remarque que l'alternance sélectionne des classes permettant de construire des arbres valides, elle borne donc le processus de croisement. Plus précisément, elle réalise la sélection des arbres correspondant exactement aux réalisations syntaxiques auxquelles le léxème peut être ancré, et coindexe les arguments sémantiques avec les indexes des noeuds de ces arbres.
Dans notre approche de hiérarchisation de l'information, une alternance est une méta-classe paramétrable contenant une conjonction de classes. Un exemple d'alternance est la classe LEXEME-AIMER(X,Y) suivante :

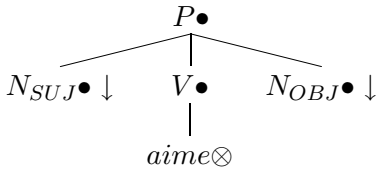
LEXEME-AIMER(X,Y)
Syntaxe : VERBE-TRANS(SUJ, OBJ) + LEMME-AIMER
Sémantique : l : aime(X,Y)
Interface : X → SUJ, Y → OBJ, Pred = *aime*.

Les classes utilisées dans cette alternance sont définies ci-dessous :

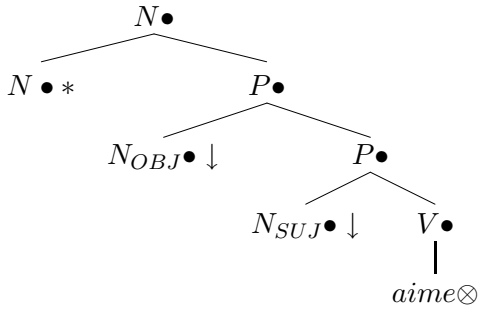
VERBE-TRANS(SUJ,OBJ)
Conjonction : SUJET(SUJ)
+ ACTIF
+ OBJET(OBJ)

LEMME-AIMER			
Disjonction :			
AIME	AIMENT	AIMAIENT	...
$V\circ$	$V\circ$	$V\circ$	
|	|	|	
$aime\otimes$	$aiment\otimes$	$aimaient\otimes$	

L'alternance LEXEME-AIMER engendre, entre autres, les arbres élémentaires suivants :



(SUJ-CAN+ACTIF+OBJ-CAN)
(Jean aime Marie)



(SUJ-CAN+ACTIF+OBJ-REL)
(La femme que Jean aime)

[55]Un léxème est une unité significative minimale non - grammaticale.

Dans notre dictionnaire, nous aurons comme entrées les alternances associées aux différents lemmes. Chaque alternance générera des arbres élémentaires et non des schèmes, puisque l'item lexical sera ancré.
On remarque que l'intégration du calcul sémantique dans une méta-grammaire à noeuds colorés en suivant l'approche utilisant la sémantique plate présentée précédemment ne pose pas de problème d'un point de vue théorique.

4.3 Application : écriture d'une petite méta-grammaire à portée sémantique

A présent, nous allons écrire une petite méta-grammaire à noeuds colorés intégrant une représentation sémantique. Avant cela, il convient de donner quelques précisions sur l'implémentation du résolveur de descriptions colorées. Ce résolveur a été programmé en langage Oz version 3. Notre choix s'est porté sur ce langage car il dispose, entre autres, des types prédéfinis *Finite Set* et *Record* permettant l'expression de contraintes sur des ensembles finis et la manipulation des structures de traits. De plus, c'est un langage libre, maintenu régulièrement, et accompagné d'un environnement de développement complet et d'une documentation abondante (voir [Oz]).

Le résolveur lit une méta-grammaire écrite dans un fichier texte. On distingue au sein de ce fichier les classes de la méta-grammaire et les entrées du dictionnaire. Une classe comprend deux dimensions : syntaxique et sémantique. Dans la première, nous définissons les descriptions de fragments d'arbres ainsi que les informations associées aux différents noeuds (catégorie, structures de traits, et type notamment) comme nous l'avons vu en 4.1.1. Dans la seconde, nous stockons la formule de sémantique plate.
Notons qu'une classe dispose à présent d'une interface (ou boîte de nommage) utilisant les fonctions grammaticales pour désigner un trait d'un noeud spécifique. Cette boîte de nommage permet la coindexation des variables de la formule sémantique. La définition d'une classe de notre méta-grammaire se fait au moyen des notations suivantes (en plus de celles présentées en 4.1.1) :

$$\text{class X} = \text{E}$$

$$\textit{où } E ::= \begin{cases} E :: I & (\textit{classe avec interface}) \\ < sem > \{...\} & (\textit{spécification de la dimension sémantique}) \end{cases}$$

Dans notre méta-grammaire, nous devons en premier lieu définir une hiérarchie d'héritage entre classes à dimension syntaxique (i.e. contenant une description de fragment d'arbre) et interface (pour la coindexation des traits sémantiques). La hiérarchie que nous allons définir permettra de construire les verbes transitifs avec objet canonique et questionné. Cette hiérarchie est représentée à la figure 4.6.

La classe SUJ-CAN se défini comme suit :

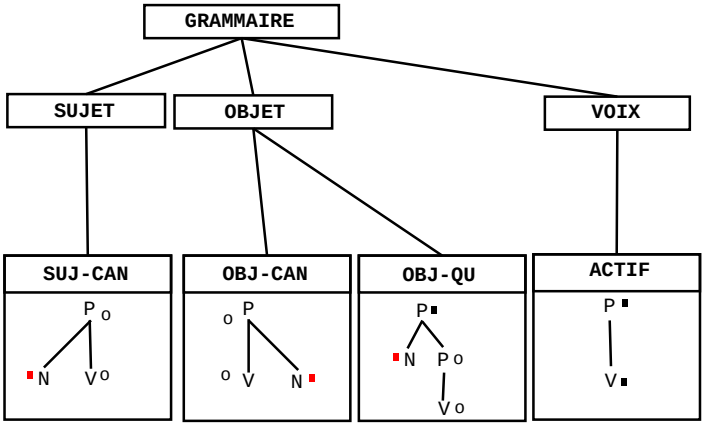


FIG. 4.6 – Petite méta-grammaire à noeuds colorés.

```
class suj-can = <syn>{
      node(white)[cat = p]{
            node(red,subst)[cat=n,idx=?W, top = [num = ?N, pers=?M]]
            node(white)[cat = v, top=[num=?N, pers=?M]]
      }
}::[suj = ?W]
```

Par rapport à la classe présentée en 4.1.1, on a ajouté une interface. Celle-ci indique que le noeud rouge de catégorie n représente la fonction grammaticale *sujet*, et établit une référence sur le trait index de ce noeud.
De plus, les classes non-feuilles de notre hiérarchie se définissent de la manière suivante :

```
class sujet = suj-can
class objet = obj-can | obj-qu
class voix  = actif
```

Ensuite, nous décrivons les classes à combiner pour obtenir les arbres des verbes transitifs :

```
class verbe-trans-actif = sujet & objet & actif
```

La prochaine étape consiste en l'écriture des classes des lemmes. Ces classes décrivent le fragment d'arbre contenant l'ancre verbale :

```
class lemmeAimer = <syn>{node(white)[cat = v]{node(red)[cat = aime]}}
| <syn>{node(white)[cat = v]{node(red)[cat = aiment]}} | ...
...
```

Il ne nous reste plus qu'à définir les alternances et à les déclarer comme entrées du dictionnaire. C'est au sein de l'alternance que l'on va utiliser la dimension sémantique :

```
class sem-trans(Pred) = <sem>{literal(!L,\$ Pred(?A,?B))
                              }::[arg0 =?A, arg1 = ?B]

class lexemeAimer = (
        verbe-trans-actif::[suj=?X,obj=?Y]
        &
        sem-trans(aimer)::[arg0 =?X, arg1 = ?Y])

value lexemeAimer
```

Nous avons ainsi défini l'alternance du verbe aimer (pour la voix active), puis nous l'avons déclarée en tant qu'entrée lexicale, ce qui permet au résolveur de construire les arbres élémentaires définis par cette alternance. Les fonctions grammaticales « suj » et « obj » de la classe *verbe-trans-actif* correspondent aux arguments du prédicat *aimer* (la coindexation de ces arguments se fait via les variables ?X et ?Y). La constante !L désigne le label de la formule de sémantique plate associée au lexème.

Nous venons de voir comment écrire une méta-grammaire à portée sémantique compilable par le résolveur de descriptions. L'intérêt principal de cet outil réside d'une part dans la facilité d'écriture et de maintenance de la méta-grammaire (cf noeuds anonymes), et d'autre part dans la puissance de calcul (liée aux mécanismes de résolution de problèmes à base de contraintes implémentés dans Oz).

Conclusion

Nous avons vu comment générer une grammaire TAG pour le français à partir d'information factorisée dans un formalisme appelé méta-grammaire. Bien que ce concept soit connu depuis le milieu des années 90, l'implémentation proposée jusqu'alors (voir [Can96]) présentait certains inconvénients concernant l'écriture et la maintenance de la méta-grammaire venant du fait que celle-ci était contrainte dans une hiérarchie tri-dimensionnelle. En conséquence, de nouvelles approches de méta-grammaires ont vues le jour, nous en avons présentées deux ici : celle du MGC reposant sur une notion de besoins/ressources permettant la factorisation des constructions syntaxiques dans une organisation pluri-dimensionnelle, et celle du résolveur de descriptions colorées où l'information syntaxique utilise des noeuds anonymes permettant une factorisation plus importante et une maintenance plus aisée.
Nous avons montré comment intégrer une représentation sémantique au processus de génération de grammaire TAG dans ces deux outils. Nous pouvons remarquer que la théorie présentée dans [GK03] y a été implémenté avec succès. Le compilateur MGC est un outil largement répandu, il a été utilisé dans le cadre de travaux utilisant différents formalismes syntaxiques (cf [KR03] et [CK03]). De plus il est multilingue (cf [Ger02]). Le résolveur de descriptions colorées représente une approche innovante, il est construit sur une logique de description d'arbres permettant une meilleure factorisation de l'information structurelle (plus de noms de noeuds à gérer). Il permet de plus une factorisation de l'information sémantique. A l'heure actuelle l'intégration de nouvelles dimensions (pour les règles générales de bonne formation des arbres TAG par exemple) sont en cours de développement.
Dans ce contexte, on peut envisager l'utilisation de logiques de description d'arbres plus expressives que celles employées actuellement, et également une évaluation de la couverture en utilisant les grammaires produites d'une part pour l'analyse syntaxique de corpus divers et d'autre part pour la génération de langue naturelle.

Remerciements

J'aimerais tout particulièrement remercier mes encadrants au sein de l'équipe Langue et Dialogue, Claire Gardent et Bertrand Gaiffe pour la qualité de leur suivi et leurs enseignements (sans oublier leurs commentaires sur ce rapport !), ainsi que Laurent Romary, responsable scientifique de l'équipe LED, pour m'avoir offert l'opportunité d'y travailler. J'aimerais également remercier les doctorants de l'équipe en les personnes de Hélène Manuélian, Benoît Crabbé et Frédéric Landragin, ainsi que mes collègues Joseph Roumier et Matthieu Quignard qui m'ont fait partager leur expérience, sans oublier l'ensemble des membres de l'équipe et du LORIA dont le contact m'a été enrichissant.
Merci à Denys Duchier grâce à qui j'ai beaucoup appris, notamment dans le domaine de la programmation par contraintes.
Enfin, je remercie le Professeur Abderrafiaa Koukam, qui fut mon suiveur de stage, de m'avoir fait connaître les possibilités offertes par le DEA, ainsi que l'ensemble des enseignants de l'Université de Technologie de Belfort - Montbéliard et du DEA IAP qui m'ont permi d'acquérir des connaissances dans des domaines variés.

Table des figures

Bibliographie

[Abe90] A. Abeillé. Lexical and syntactic rules in a tree adjoining grammar. In *Proceedings of the 28th Meeting of the Association for Computational Linguistics (ACL), Pittsburgh*, 1990.

[Abe93] A. Abeillé. *Les Nouvelles Syntaxes - Grammaires d'unification et analyse du français*. Editions Armand Colin, Linguistique, 1993.

[ACK99] A. Abeillé, M.H. Candito, and A. Kinyon. Ftag : current status and parsing scheme. In *VEXTAL, Venice, Italy*, 1999.

[BB99] P. Blackburn and J. Bos. Representation and inference for natural language : A first course in computational semantics, 1999. Available at http ://www.comsem.org.

[Bos95] J. Bos. Predicate logic unplugged. In *Proceedings of the Tenth Amsterdam Colloquium, ILLC/Department of Philosophy, University of Amsterdam, Amsterdam, Holland, December*, 1995.

[Can96] M.H. Candito. A principle-based hierarchical representation of ltags. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING'96), Kopenhagen*, 1996.

[Can99] M.H. Candito. *Représentation modulaire et paramétrable de grammaires électroniques lexicalisées : application au français et à l'italien*. PhD thesis, Université Paris 7, 1999.

[CFS99] A. Copestake, D. Flickinger, and I. Sag. Minimal recursion semantics : An introduction. Draft. Available at http ://www-csli.stanford.edu/~aac/papers/newmrs.pdf, 1999.

[CK03] L. Clement and A. Kinyon. Generating lfgs with a metagrammar. In *Proceedings of the 8th International Lexical Functional Grammar Conference, Saratoga Springs, NY*, 2003.

[Cra03] B. Crabbé. Metagrammar with lexical rules. Ebauche, 2003.

[DG99] Denys Duchier and Claire Gardent. A constraint-based treatment of descriptions. In *Third International Workshop on Computational Semantics (IWCS-3), pp. 71-85, Tilburg, NL*, 1999.

[DGN02] D. Duchier, C. Gardent, and J. Niehren. Concurrent constraint programming in oz for natural language processing, 2002. Lecture Notes. Available at http ://www.ps.uni-sb.de/~niehren/Web/Vorlesungen/Oz-NL-SS01.

[DHS+00] C. Doran, B. Hockey, A. Sarkar, B. Srinivas, and F. Xia. Evolution of the xtag system. University of Pennsylvania, 2000.

[DN00] Denys Duchier and Joachim Niehren. Dominance constraints with set operators. In *Proceedings of the First International Conference on Computational Logic (CL2000)*, volume 1861 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2000.

[DT99] Denys Duchier and Stefan Thater. Parsing with tree descriptions : a constraint-based approach. In *Sixth International Workshop on Natural Language Understanding and Logic Programming (NLULP'99), Las Cruces, New Mexico*, 1999.

[Duc99] D. Duchier. Set constraints in computational linguistics - solving tree descriptions. In *Workshop on Declarative Programming with Sets (DPS'99), Paris, pp. 91 - 98*, 1999.

[Duc00] D. Duchier. Constraint programming for natural language processing, 2000. Lecture Notes, ESSLLI 2000. Available at http ://www.ps.uni-sb.de/Papers/abstracts/duchier-esslli2000.html.

[GCR02] B. Gaiffe, B. Crabbé, and A. Roussanaly. A new metagrammar compiler. In *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6), Venice*, 2002.

[Ger02] K. Gerdes. Dtag ? - attempt to generate a useful tag for german using a metagrammar. In *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6), Venice*, 2002.

[GK03] C. Gardent and L. Kallmeyer. Semantic construction in ftag. In *Proceedings of the 10th meeting of the European Chapter of the Association for Computational Linguistics, Budapest*, 2003.

[Gro01] XTAG Research Group. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania, 2001. Available at http ://www.cis.upenn.edu/~xtag/gramrelease.html.

[Jos87] A. Joshi. Unification and some new grammatical formalisms. In *Proceedings of the Theoretical Issues in Natural Language Processing 3 (TINLAP 3), Las Cruces, NM, pp. 45 - 50*, 1987.

[JS97] A. Joshi and Y. Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69 – 124. Springer, Berlin, New York, 1997.

[KJ85] A. Kroch and A. Joshi. The linguistic relevance of tree adjoining grammars. Technical report, MS-CIS-85-16, University of Pennsylvania, Philadelphia, 1985.

[KN00] A. Koller and J. Niehren. Constraint programming technology in computational linguistics. Technical report, Universitat des Saarlandes, Programming Systems Lab. Submitted. Available at http ://www.ps.uni-sb.de/Papers/abstracts/CP-NL.html, 2000.

[KR03] A. Kinyon and O. Rambow. The metagrammar : a cross-framework and cross-language test-suite generation tool. In *LINC-EACL-2003, Budapest*, 2003.

[MT90] P. Miller and T. Torris. *Formalismes syntaxiques pour le traitement automatique du langage naturel.* Editions HERMES, 1990.

[Oz] Mozart Oz. The oz mozart consortium. http ://www.mozart-oz.org.

[RVS92] J. Rogers and K. Vijay-Shanker. Reasoning with descriptions of trees. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics, pp. 72 - 80*, 1992.

[RVS94] J. Rogers and K. Vijay-Shanker. Obtaining trees from their descriptions : An application to tree-adjoining grammars. *Computational Intelligence, 10 :401–421*, 1994.

[VS92] K. Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics, vol.14, num.4 :481–517*, 1992.

[VSJ88] K. Vijay-Shanker and A. Joshi. Feature structures based tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING'88), Budapest, pp. 714 - 719*, 1988.

[VSS92] K. Vijay-Shanker and Y. Schabes. Structure sharing in lexicalized tree adjoining grammars. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING'92), Nantes, pp. 205 - 212*, 1992.

Résumé

Ce travail présente le concept de méta-grammaire qui est à la base du procédé de production semi-automatique de grammaires d'arbres adjoints. Les divers approches de méta-grammaire, notamment dans le cadre de la génération de grammaires à large couverture pour le français, sont étudiées. Un accent particulier est mis sur l'intégration d'une représentation sémantique dans deux implémentations de méta-grammaire : le compilateur MGC et le résolveur de descriptions colorées. Cette intégration s'appuit sur les travaux menés au sein de l'Action de Recherche Concertée INRIA «GenI» (Génération et Inférence).

Mots-clés: grammaires d'arbres adjoints, méta - grammaire, compilation.

Abstract

In our work, we aimed at introducing the concept of meta-grammar, which makes possible the production of wide-coverage Tree Adjoining Grammars. Different approaches have been studied, especially in the area of semi-automatic generation of wide-coverage TAG for French. In this report, we focus on the integration of semantic calculus into two implementations of meta-grammar compilers : the MGC and the coloured descriptions-based Solver. This work follows the ideas developped in the "GenI" project (Action de Recherche Concertée INRIA "Génération et Inférence").

Keywords: Tree Adjoining Grammars, Metagrammar, compilation.