Transfert et génération de phrases

Sujiva Pinnagoda et Mira Ramluckun
Département de linguistique - LATL
Université de Genève

septembre 1993

1 Introduction

Ce document décrit les modules de transfert et de génération développé pour le système ITS-2. Ces modules interviennent une fois que l'analyseur syntaxique a segmenté la phrase source en syntagmes. Ces syntagmes sont représentés par une structure de donnée arborescente. L'assimilation de cette structure est importante pour la compréhesion de l'approche utilisée pour le transfert et la génération des phrases. Prenons une phrase simple et examinons son analyse:

(1)a. John likes Mary.

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ John][$_{T'}$ likes [$_{DP}$ Mary]]]]

Pour des raisons d'affichage à l'écran, nous utiliserons la représentation de la structure arborescente en 1(b) au lieu de celle en 1(c):

(1)c

- la branche de gauche de l'arbre est appelée le spécificateur
- la branche de droite, le complément
- le mileu, la tête

Lors de l'affichage, nous écrivons tout d'abord la catégorie de l'arbre (ou sous-arbre), ensuite le spécificateur qui est une liste tout comme le complément, c'est-à-dire qu'elle peut contenir plusieurs syntagmes. Pour différencier entre les éléments du spécificateur et la tête, on ouvre une parenthèse carrée; comme on ne peut avoir qu'un seul syntagme comme tête, suivi de son complément, celà nous donne :

```
[cat [spécificateur] [cat' tête [complément]]
```

Ceci n'est qu'un petit aperçu de la structure complexe utilisée dans ce projet (cf. FG-BTOOLS.DEF et GBTOOLS.DEF). Une fois que l'analyse a été appliquée, on peut commencer les opérations de transfert et de génération de phrases. Lors de la traduction, nous gardons au tant que possible la forme syntaxique ainsi que le style de la langue source. En vertu de ce principe, si une phrase est au passif, on cherchera à maintenir cette construction dans la phrase cible.

L'analyseur n'accepte que des phrases complètes (CP) et des syntagmes nominaux, notés comme des (DP). Il n'y a pas de différence fondamentale entre le transfert des phrases et cela des syntagmes nominaux, car le transfert des DP est un sous-ensemble du transfert des CP.

Dans ce document nous allons expliquer les algorithmes utilisés. Le deuxième chapitre explique en détail toutes les étapes nécessaires pour le transfert et la génération des phrases simples et aussi des syntagmes nominaux, qui demandent des traitements spéciaux, comme c'est le cas pour le possessif, le génitif et le déterminant vide. Le troisième chapitre est consacré aux autres formes de générations de phrases disponibles dans ITS-2.

2 Le transfert et génération de cas simple

L'algorithme général utilisé est une descente récursive, qui consiste à chercher l'élément clé (ou la tête sémantique) du syntagme, ensuite à parcourir le spécificateur suivi du complément. Dans le cas des phrases complètes, ceci consiste à trouver le verbe principal de la proposition et, dans le cas des syntagmes nominaux, de trouver la tête nominale du syntagme.

2.1 Les phrases simple

Le point de départ pour la traduction d'une phrase est de trouver le verbe principal, car les traits lexicaux du verbe cible permet de générer la structure cible adéquate.

Les étapes importantes pour le transfert d'une phrase simple sont les suivantes:

1. création du CP **TranslateCP**
2. recherche des verbes **GetMainVerb**
3. transfert de la tête du verbe principal **TranslateVerbHead**
4. transfert des arguments **Trarguments**

5. application des transformations **ApplyTransf**

6. génération morphologique **ApplyMorpho**

7. création de la phrase cible **StandardWriteList**

Chacune de ces étapes à être élaborée dans les sous sections suivantes. Ces étapes sont similaires pour les deux traductions: anglais/français ou français/anglais. Lorsqu'une différente approche est utilisée, elle est détaillée explicitement.

2.1.1 Création du CP

Dans le cas d'une phrase simple, le CP a une tête vide et on le traite comme un transfert de syntagme nominal; il s'agit de trouver la tête du syntagme. Dans ce cas précis, la tête est vide, donc on doit créer un nœud cible identique (contenant les mêmes traits) au nœud source. Une fois que ce nœud a été créé, il faut parcourir le spécificateur et le complément du nœud source, de manière récursive. Pour une phrase simple, le spécificateur est une liste vide, mais par contre le complément ne l'est pas. Il contient justement le verbe, qui peut soit être le verbe principal soit un auxiliaire; donc, on fait appel au traitement des verbes **TranslateVerbNode**.

2.1.2 Recherche des verbes

La recherche des verbes consiste à parcourir l'arbre en profondeur, en prennant toujours la branche de droite (le complément), car elle est la seule susceptible de contenir un verbe. Le verbe principal est le dernier verbe (VP) de la proposition. De ce fait, il faut parcourir l'arbre jusqu'à la fin. Dans certaines phrases comme

(2) John is eating.

le verbe principal ne contient pas le temps, mais c'est l'auxiliaire qui le contient. L'auxiliaire est le premier verbe (TP) de la proposition, sauf pour les phrases interrogatives où il se trouve dans la tête du complémenteur (CP).

(3)a. Is John eating

b. [$_{CP}$ is [$_{TP}$ [$_{DP}$ John] [$_{T'}$ [$_{VP}$ eating]]]]

Les verbes source étant trouvés, grâce à la procédure **GetMainVerb**, on doit chercher le verbe cible correspondant au verbe principal. Il faut noter que les auxiliaires ne sont jamais transférés, mais construits en fonction des besoins. Le verbe cible doit avoir les mêmes traits lexicaux que le verbe source **TranslateVerbHead**

2.1.3 Transfert de la tête du verbe

Le transfert d'un verbe est plus complexe que les autres syntagmes, étant donné le fait qu'il est toujours associé à plusieurs lexèmes. Déjà au niveau source il faut sélectionner le bon lexème source pour pouvoir trouver le bon verbe cible correspondant **MatchWithLexeme**.

Correspondance lexicale du verbe principal source

L'analyseur syntaxique, lorsqu'il trouve un verbe, donne tous ses homonymes. Il faut alors trouver le lexème qui permet de faire la correspondance entre les arguments du verbe et les arguments du lexème. Néanmoins, la correspondance stricte entre les arguments n'est pas obligatoire (le nombre d'arguments du lexème ne doit pas être strictement égal au nombre d'arguments du verbe). Pour chaque homonyme, on va faire la correspondance entre les fonctions grammaticales des arguments du verbe et celles du lexème. Si on n'y arrive pas, on prend le lexème suivant. Une fois que le lexème source trouvé, il faut trouver le lexème cible correspondant **SelectTargetItem**.

Sélection du lexème cible

Dans le dictionnaire bilingue on va chercher tous les lexèmes cible correspondant au lexème source sélectionné. Mais lors du transfert, on ne veut qu'un seul lexème cible. De ce fait, s'il y a plus d'une correspondance et si le mode interactif a été choisi, alors l'utilisateur doit choisir le terme le plus adéquat; dans le cas contraire, le premier lexème est choisi par défaut. Cette sélection est identique dans le cas des syntagmes nominaux. Il faut maintenant trouver les correspondances entre les fonctions grammaticales du lexème cible et les arguments du verbe source **MapTargetLexeme**. Ces correspondances sont répertoriées dans un tableau de type **TargetMapTable**. Le verbe cible étant trouvé, on peut transférer ses arguments **Trarguments**.

2.1.4 Transfert des arguments

L'ordre de transfert des arguments dépend des fonctions grammaticales du verbe cible et non du verbe source. Les arguments transférés sont mis dans la table d'arguments du verbe principal cible avec leurs fonctions grammaticales. Les fonctions grammaticales traitées sont :

- le sujet est toujours le premier élément à être traduit, sauf lorsque le verbe possède le trait **non_theta_subject**, dans quel cas il ne sera pas traduit à ce niveau. On traitera ce cas ultérieurement.
- seul le complément du prép_objet est transféré, c'est-à-dire la préposition n'est pas traduite car la correspandance directe n'est pas forcément la plus adéquate. Le lexème contient la bonne préposition que l'on extrait **ExtractPrep** et ajoute au nœud transféré **MakeVerbPP**.
- l'objet_direct est transféré sans restriction.
- le pred_objet sera discuté lorsqu'on traitera les phrases causatives et les phrases prédicatives.
- le S_objet implique le transfert d'une phrase enchassée. Le traitement de cette fonction est discuté plus tard.

Le transfert des arguments nous donne la structure profonde[1] de la phrase cible. Comme le système veut que les phrases cibles conservent la structure source, on applique les transformations nécessaires sur la structure profonde cible. Dans le cas d'une phrase simple, il n'y a pas de transformation spécifique, sauf bien sûr l'application du temps **ApplyTransf**.

2.1.5 Application des transformations

Pour une phrase simple, la seule transformation est le transfert du temps **TenseTransfer**. Les autres transformations sont expliquées dans le troisième chapitre.

Il faut convertir le temps source en temps cible. Le temps cible est déterminé par deux traits du verbe source, le premier étant **tense_s**, le second **v_features**. De plus, en français il faut tenir compte du mode (indicatif, conditionnel, etc.). La conversion est résumée dans la table ci_dessous:

source			
		tense_s	v_features
cible			
mode	tense_s		
indicatif	infinitif passé	infinitive	perfective
indicatif	infinitif	infinitive	
indicatif	passé composé	present	perfective
indicatif	présent	present	
indicatif	plus-que-parfait	past	perfective
indicatif	imparfait	past	
indicatif	participe passé	past_participle	
indicatif	future antérieur	future	perfective
indicatif	future simple	future	
conditionnel	passé composé	conditional	perfective
conditionnel	présent	conditional	

Le temps est choisi selon cette table. Dans le cas d'un temps composé, il faut ajouter l'auxiliaire **avoir** ou **être**. Dans le lexème du verbe, il y un trait indiquant lequel des deux auxiliaires doit être utilisé. Cet auxiliaire n'est pas recherché dans le lexique mais il est cré **CreateNode** et ajouté devant le verbe principal (comme nœud mère du verbe principal). Pour l'instant, les arguments sont dans la table d'arguments. Ensuite, il faut créer l'arbre cible. Le sujet est toujours ajouté dans le spécificateur du premier verbe et le reste des arguments dans le complément du verbe principal **InitVerbCompl**. A noter que le seul cas où le verbe principal n'est pas le premier verbe, c'est lorsque le temps de la phrase est composé, comme le passé composé.

[1] C'est une représentation sous forme de structure canonique qui montre les relations entre les prédicats et les arguments.

Maintenant que les arguments ont été traités, il faut parcourir le spécificateur et le complément du verbe principal **TraverseVerb**. Mais, dans le cas d'une phrase simple, ces listes sont vides.

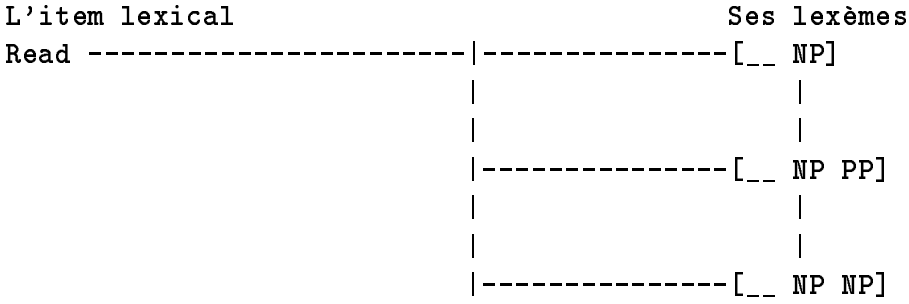
Lorsqu'on a transféré, on n'a gardé que la forme de base des syntagmes. Par exemple, si on a transféré **the cats**, on obtient la correspondance de base **les chat**. Comme le nœud **chat** contient le trait pluriel, il faut aller chercher dans le lexique le bon lexème **ApplyMorpho**.

2.1.6 La génération morphologique

La génération morphologique est effectuée de deux manières. Dans le cas géneral, on recherche la forme adéquate sur la base du numéro de lexème et des traits lexicaux dans le lexique. Pour un nombre limité de cas (notamment dans le cas des pronoms), on modifie uniquement la chaîne de caractères. Plus tard, il faudrait la modifier.

La transformation morphologique doit être appliquée à tous les nœuds de l'arbre cible. La recherche de la forme morphologique est la même pour les NP, les AP et les VP.

Pour comprendre l'algorithme, il faut d'abord comprendre la structure du lexique **FLRecords.Def**. Si on regarde la structure, on voit qu'un item lexical peut être associé à plusieurs lexèmes. L'item lexical contient la forme de base. Prenons l'exemple du verbe **read**:



Les lexèmes sont reliés entre eux par le pointeur **homo**, et tous ces lexèmes pointent vers l'item lexical qui est la forme de base. De plus, toutes les formes morphologiques pointent vers le premier lexème. Tous ces lexèmes ont le même word_index (celui de la forme de base) et toutes les formes morphologiques ont le même lexeme_index.

Lorsqu'on effectue la recherche d'une forme morphologique, on a en entrée le lexème avec les traits lexicaux désirés. Le lexème nous permet de trouver l'item lexical, ce qui nous donne la forme de base. Cette dernière pointe vers le premier lexème, le quel possède toutes les formes morphologiques. Parmi ces formes, on choisit la forme adéquate selon les traits lexicaux désirés. On peut résumer l'algorithme de la manière suivante:

- on réordonne le fichier des mots français avec la clé word_index **Resetk**
- on se positionne sur la forme de base du mot avec le word_index **Fetchk**
- on réordonne le fichier des mots français avec la clé lexeme_index; ainsi, on a le premier lexème
- on parcourt la liste des mots qui ont le même lexeme_index jusqu'à l'obtention de la forme morphologique voulue.

La phrase cible est construite. Mais, comme elle est sous forme d'arbre, il faut l'écrire sous forme de chaîne de caractères **StandardWriteList** pour pouvoir l'afficher à l'écran.

2.1.7 La création de la phrase cible

On crée la chaîne de caractères à partir du champs **key**, qui est dans le champs **head** du nœud (ProjectionPtr) **InitTargetSentence**. Pour cela, il faut parcourir l'arbre cible en passant d'abord par le spécificateur, puis par la tête et le complément.

Un espace est toujours mis entre les mots, sauf au début d'une phrase ou lorsqu'il y a une élision, par exemple dans **j'ai**. L'élision affecte certains items fonctionnels, comme les déterminants, les clitiques et les complémenteurs monosyllabiques qui se terminent par "a" ou "e". Si le mot suivant commence par une voyelle, on remplace la dernière lettre de l'item fontionnel par une apostrophe. Le test d'élision est fait dans **CheckElision** et le remplacement est effectué dans **InitTargetSentence**. Lors la procédure du test d'élision, on traite aussi les cas suivants:

- celui des déterminants démonstratifs, où le masculin singulier prend un "t" devant une voyelle. Ceci est traité comme une élision, mais au lieu de remplacer la dernière lettre, on ajoute un "t" **InitTargetSentence**
- celui des déterminants possessifs, où le féminin singulier est écrit comme le masculin singulier devant une voyelle **InitTargetSentence**.

Il existe encore à présenter le test de la contraction **ApplyContraction**, qui est:

- si on a une préposition de type "de" suivie d'un déterminant défini
  - "le", alors la préposition devient "du"
  - "les", alors la préposition devient "des"
- si on a une préposition de type "à" suivie d'un déterminant défini
  - "le", alors la préposition devient "au"
  - "les", alors la préposition devient "aux"

Une fois que la chaîne de caractères est formée, on l'affiche à l'écran. On a vu toutes les étapes nécessaires au transfert d'une phrase simple. La prochaine section explique les étapes pour le transfert des syntagmes nominaux qui ont une structure particulière.

2.2 Les syntagmes nominaux

Le transfert des syntagmes nominaux est déclenché par la procedure **TranslateHead**. Le point de départ est de transférer la tête du syntagme, ensuite le spécificateur et finalement le complément. Mais, dans le cas des possessifs, des génitifs et des déterminants vides, le premier syntagme est une tête vide. Donc il n'y a pas de point de départ. De plus, leur structure source est différente de leur structure cible. Donc, on ne peut pas faire un transfert normal. Il faut plutôt construire la structure cible.

2.2.1 Les syntagmes possessifs

Anglais/français La forme syntaxique du syntagme possessif source (anglais) en 4(b) différe quelques peu des syntagmes ordinaires. La structure cible (français) du syntagme possessif est representée en 4(c):

(4)a. my dog.

b. [$_{DP}$ [$_{DP}$ my] [$_{D}$, [$_{NP}$ dog]]]

c. [$_{DP}$ mon[$_{NP}$ chien]]

Si on compare les deux structures, on voit que le spécificateur du syntagme principal DP source est non vide, par contre, sa tête est vide. Dans la structure cible, le spécificateur du syntagme principal DP est vide, mais, par contre, la tête est non vide. L'esquisse de l'algorithme de **CreatePossesifDet** est la suivante:

- création du nœud possessif "son" **CreateNode**
- extraction du spécificateur du nœud vide source
- mise à jour des traits du nœud possessif cible selon le spécificateur
- extraction du complément du nœud vide source
- transfert de ce nœud
- attachement de ce dernier comme le complément du nœud possessif cible.

Pour tous les possessifs on crée le déterminant possessif "son", puis on modifie ses traits. Lors de la transformation morphologique **GetDetForm**, on change la chaîne de caractères qui est dans le champs key du pointeur head du nœud possessif.

2.2.2 Les syntagmes génitifs

Anglais/français La forme syntaxique des syntagmes génitifs de l'anglais est différente de celle du français. Donc, il faut également faire une reconstruction.

(5)a. John's dog

b. [$_{DP}$ [$_{DP}$ John] [$_{D}$, [$_{NP}$ dog]]]

c. [$_{DP}$ le[$_{NP}$ chien[$_{PP}$ de[$_{DP}$ John]]]]

Il faut noter qu'on peut avoir des possessifs imbriqués dans les génitifs:

(6)a. my friend's dog

b. [$_{DP}$ [$_{DP}$ [$_{DP}$ my] [$_{D}$, [$_{NP}$ friend]]] [$_{D}$, [$_{NP}$ dog]]]

c. [$_{DP}$ le[$_{NP}$ chien[$_{PP}$ de[$_{DP}$ mon[$_{NP}$ ami]]]]]

ou encore:

(7)a. my sister's friend's dog

b. [$_{DP}$ [$_{DP}$ [$_{DP}$ [$_{DP}$ my] [$_{D}$, [$_{NP}$ sister]]] [$_{D}$, [$_{NP}$ friend]]] [$_{D}$, [$_{NP}$ dog]]]

c. [$_{DP}$ le[$_{NP}$ chien[$_{PP}$ de[$_{DP}$ le[$_{NP}$ ami[$_{PP}$ de [$_{DP}$ ma[$_{NP}$ soeur]]]]]]]]

Le principe est très similaire à celui décrit précédemment. Mais, l'approche est récursive **CreateSaxonDet**. On peut voir que le complément source prend la position du spécificateur cible. De ce fait, il faut commencer par le transfert du complément.

Remarquons que le déterminant source "the" correspond aux déterminants définis "le", "la" et "les". Donc, lorsqu'on doit transférer un déterminant source ou qu'on doit créer un déterminant défini, il faut qu'on ait ces trois articles. Dans ce cas, on crée un nœud où le pointeur **head** contient ces trois articles **CreateNode("the", FGBTools.D, t_c)**. Ceci est vrai pour tous les syntagmes nominaux contenant le déterminant "the". L'esquisse ci-dessous décrit l'algorithme des syntagmes génitifs:

- création du déterminant défini
- extraction et transfert du complément source
- extraction du spécificateur source
- création de la préposition "de"
- transfert du spécificateur avec éventuellement un appel récursif ou un appel à la procédure CreatePossesifDet.

2.2.3 Les déterminants vides

Anglais/français En anglais le déterminant peut être omis dans certain cas, contrairement au français:

(8)a. apples

b. [$_{DP}$ [$_{NP}$ apples]]

c. [$_{DP}$ [$_{PP}$ des [$_{DP}$ [$_{NP}$ pommes]]]]

Lorsqu'en anglais il n'y a pas de déterminant, on crée un déterminant indéfini qui peut être soit "de" soit "du" soit "des" selon le nom. L'approche utilisée est la suivante:

- création d'un nœud DP vide identique au source
- extraction et transfert du complément source
- création du déterminant indéfini et le projette comme une préposition PP
- si le nœud transféré est
  - masculin ou pluriel, alors création d'un DP vide comme précédemment

- feminin, alors création de l'article défini "la"

- au premier nœud DP vide on attache d'abord
  - le nœud prépositionnel PP
  - soit le deuxième DP vide soit l'article défini "la"
  - le nœud transféré

3 Le transfert et génération des cas plus complexes

Le transfert des phrases plus complexes demande l'application des transformations au niveau de la structure profonde et même parfois une reconstruction totale des structures cibles, comme c'est le cas des modaux, avec lasquels il est impossible de préserver la structure source. Dans cette section, on va expliquer les algorithmes utilisés pour ces transformations ainsi que pour les restructurations.

3.1 Les phrases passives

La construction d'une phrase passive est une transformation simple qu'on applique sur la structure profonde. Le transfert se déroule comme pour une phrase simple jusqu'au transfert des arguments **Trarguments**

La particularité d'une phrase passive est la montée de l'objet_direct à la position du sujet. Si un sujet logique est realisé il occupe la position de complément accompagné de la préposition "par".

En examinant la structure syntaxique des phrases passives on voit bien la montée de l'objet_direct, car il est répresenté par une trace[2]. De plus, les phrases passives utilisent l'auxiliaire "être".

(9)a. the mouse has been eaten.

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ the[$_{NP}$ mouse]]$_i$[$_{T}$, has[$_{VP}$ been[$_{VP}$ eaten[$_{DP}$ e]$_i$]]]]]

c. [$_{CP}$ [$_{TP}$ [$_{DP}$ la[$_{NP}$ souris]]$_i$[$_{T}$, a[$_{VP}$ été[$_{VP}$ mangée[$_{DP}$ e]$_i$]]]]]

(10)a. the mouse has been eaten by the cat.

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ the[$_{NP}$ mouse]]$_i$[$_{T}$, has[$_{VP}$ been[$_{VP}$ eaten[$_{DP}$ e]$_i$[$_{PP}$ by[$_{DP}$ the[$_{NP}$ cat]]]]]]]

c. [$_{CP}$ [$_{TP}$ [$_{DP}$ la[$_{NP}$ souris]]$_i$[$_{T}$, a[$_{VP}$ été[$_{VP}$ mangée[$_{DP}$ e]$_i$[$_{PP}$ par[$_{DP}$ le[$_{NP}$ chat]]]]]]]

L'esquisse de cet algorithme **ApplyPassive** qui est dans la procédure **ApplyTransf** est le suivant:

[2] C'est un nœud vide représentant un nœud non-vide plus haut dans l'arbre.

- si le sujet est non vide alors
  - créer la préposition "par"
  - ajouter le sujet comme le complément de "par"
  - trouver la première position vide de la table d'arguments du verbe principal et y mettre le nœud "par" (donc le nœud sujet) avec la fonction grammaticale "sujet"
- prendre l'objet_direct et le mettre à la position sujet, donc au début de la table d'arguments
- créer la trace de l'objet_direct et le mettre dans la table d'arguments à la position de l'objet_direct
- créer l'auxiliaire être et y ajouter le verbe principal comme son complément
- metter le verbe principal au participe passé

Le reste du transfert est identique à une phrase simple.

3.2 Les constructions infinitives

La particularité des phrases infinitives est qu'elles sont au moins bi-propositionnelles. Mais on commence le traitement comme pour les phrases simples, c'est-à-dire par le transfert des arguments. La seule différence est qu'on a une autre phrase comme argument. A noter qu'il est possible d'avoir plusieurs phrases enchassées. Puisqu'on traite toujours les arguments en premier, on va traiter d'abord la phrase enchassée et ensuite la phrase principale.

(11)a. John likes to eat.

b. $[_{CP}\ [_{TP}\ [_{DP}$ John$]_i[_{T}$, likes$[_{CP}\ [_{TP}\ [_{DP}$ e$]_i[_{T}$, to$[_{VP}$ eat$]]]]]]]$

c. $[_{CP}\ [_{TP}\ [_{DP}$ John$]_i[_{T}$, aime$[_{CP}\ [_{TP}\ [_{DP}$ e$]_i[_{T}$, $[_{VP}$ manger$]]]]]]]$

Il y a deux types de construction pour des phrases infinitives:

- des constructions à montée
- des constructions à contrôle qui peut être soit
  - contrôle de sujet
  - contrôle d'objet_direct

3.2.1 Construction à montée

Pour qu'une phrase soit une construction à montée, il faut que le verbe principal de la première proposition, qu'on appelle le verbe gouverneur, soit un verbe à montée, comme **seem**, **appear**, etc.

(12)a. John seems to sleep

b. $[_{CP}\ [_{TP}\ [_{DP}$ John$]_i[_{T}$, seems$[_{CP}\ [_{TP}\ [_{DP}$ e$]_i[_{T}$, to$[_{VP}$ sleep$]]]]]]]$

c. [$_{CP}$ [$_{TP}$ [$_{DP}$ John]$_i$[$_{T}$, semble[$_{CP}$ [$_{TP}$ [$_{DP}$ e]$_i$[$_{T}$, [$_{VP}$ dormir]]]]]]]

On commence le transfert comme pour une phrase simple, c'est-à-dire par le transfert des arguments. Une particularité est que l'argument sujet est marqué du trait non_theta_subject; donc il ne sera pas transféré à ce niveau, mais plutôt au moment où l'on transfère les arguments de la phrase enchassée. Quant à l'autre argument que contient la phrase enchassée, sa fonction grammaticale est S_object. Avant de traiter la phrase enchassée, on sauvegarde le verbe gouverneur dans la variable globale **infoVal.govern**. Ensuite, on commence le transfert de la phrase enchassée. Une fois que les arguments sont transférés on applique la construction infinitive. L'algorithme pour les verbes à montée est le suivant:

- on prend le sujet du verbe principal de la phrase enchassée et on le met dans la table d'arguments du verbe gouverneur (car le sujet n'a pas été transféré au niveau de la phrase principale)
- on crée sa trace et on le met dans la table d'arguments du verbe principal de la phrase enchassée.

Le verbe enchassé est toujours à la infinitif et le reste du traitement est semblable aux phrases simples. Une fois qu'on a traitée la phrase enchassée on revient sur la phrase principale.

3.2.2 Construction à contrôle

On a traité deux types de contrôle. Le premier exemple est un contrôle de sujet et le deuxième un contrôle d'objet.

(13)a. John promises Mary to come.

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ John]$_i$[$_{T}$, promises[$_{DP}$ Mary] [$_{CP}$ [$_{TP}$ [$_{DP}$ e]$_i$[$_{T}$, to[$_{VP}$ come]]]]]]]

c. [$_{CP}$ [$_{TP}$ [$_{DP}$ John]$_i$[$_{T}$, promet[$_{PP}$ à[$_{DP}$ Mary]][$_{CP}$ de[$_{TP}$ [$_{DP}$ e]$_i$[$_{T}$, [$_{VP}$ venir]]]]]]]

(14)a. he persuades Mary to come.

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ John] [$_{T}$, persuades[$_{DP}$ Mary]$_i$ [$_{CP}$ [$_{TP}$ [$_{DP}$ e]$_i$[$_{T}$, to[$_{VP}$ come]]]]]]]

c. [$_{CP}$ [$_{TP}$ [$_{DP}$ John] [$_{T}$, persuade[$_{DP}$ Mary]$_i$ [$_{CP}$ de[$_{TP}$ [$_{DP}$ e]$_i$[$_{T}$, to[$_{VP}$ venir]]]]]]]

L'algorithme pour les phrases avec contrôle de sujet:

- on compare si le sujet du verbe gouverneur et celui du verbe principal de la phrase enchassée est identique
- si oui, on crée la trace du sujet et on la met dans la table d'arguments du verbe principal à la position sujet
- sinon, la phrase ne peut pas être infinitive

Par exemple:

(15)a. John veut venir

b. John veut que Mary vienne

L'algorithme pour les phrases avec contrôle d'objet est identique, sauf qu'on compare les objets_directs.

Ces algorithmes de constructions infinitives sont dans la procédure **ProcessInfinitif**, qui est une procédure booléan. Elle retourne TRUE lorsqu'elle a pu appliquer un des algorithmes de constructions infinitives et bien sûr FALSE dans le cas contraire.

Les exemples ci-dessous montre que certaine phrases cibles ne peuvent pas être infinitives, mais doivent être conjugées. Dans le cas des phrases conjuguées, il faut encors ajouter "que" dans la tête du complémenteur (CP). La phrase est conjugée selon le verbe gouverneur. Il se peut même qu'on doit changer le mode (subjonctif, conditionnel).

(16)a. John wants Mary to come

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ John] [$_{T}$, wants[$_{CP}$ [$_{TP}$ [$_{DP}$ Mary] [$_{T}$, to[$_{VP}$ come]]]]]]]

c. [$_{CP}$ [$_{TP}$ [$_{DP}$ John] [$_{T}$, veut[$_{CP}$ que[$_{TP}$ [$_{DP}$ Mary] [$_{T}$, vienne]]]]]]

Certaines phrases cibles infinitives exigent une préposition comme "à" ou "de" dans la tête du complémenteur de la phrase enchassée. Le choix de cette préposition et le changement du mode se trouvent dans la structure argumentale du lexème du verbe gouverneur.

Il faut noter que certaines phrases multi-enchassées infinitives, où la phrase cible ne peut pas être infinitive, doit être conjugées selon le verbe gouverneur. De ce fait, il est important de sauvegarder le temps dans une variable globale **infoVal.tense**.

(17) he seems to want Paul to read the book

Cette variable est mise à jour lors de la descente récursive des phrases enchassées et on ne la met à jour que si l'on trouve un verbe gouverneur autre qu'infinitif.

Les exemples précédents nous montrent que toutes les phrases source infinitives ne peuvent pas être traduites comme des infinitives. Il est donc important de faire un certain nombre de tests. On a traité les phrases interrogatives indirectes avec les phrases infinitives, car leur traitement est semblable.

(18)a. he wonders whether to go.

La procédure **InitTense** qui se trouve dans ApplyTransf fait justement ces tests. Cette procédure booléaenne n'est activée que lors des transferts des phrases enchassées; elle retourne TRUE dans ce cas là et FALSE lors du transfert des phrases non-enchassées où le transfert du temps **TenseTransfer** est activé. L'algorithme de **InitTense** est le suivant:

- quand la phrase enchassée est infinitive, on teste
  - si la phrase est interrogative (contient le trait wh_comp) et si , en plus, la tête du CP est différente de NIL. Dans ce cas, on crée une phrase interrogative cible en mettant dans la tête du complémenteur cible "si". Puisque la phrase doit être

conjugée, elle sera conjuguée selon le verbe gouverneur; si la tête est NIL, mais contient quand même le trait interrogatif, on construit une phrase interrogative cible dont la tête du CP est vide. Plus tard, l'élément_wh sera projeté dans le spécificateur du CP (cf. le traitement du mouvement_wh)

- sinon, si le lexème du verbe gouverneur cible accepte la forme infinitive, alors on applique la procedure **ProcessInfinitif**. Si la procedure retourne :
  * TRUE, alors la phrase enchassée est infinitive, et on teste si le verbe gouverneur a besoin d'une préposition. Si oui, on la crée;
  * FALSE, alors la phrase enchassée est conjugée selon le verbe gouverneur, et on vérifie s'il faut changer de mode. Si le mode change, la phrase reste au présent. Par exemple, si le mode devient subjonctif, alors la phrase sera au subjontif présent

- si la phrase enchassée est conjugée, on applique les mêmes tests qu'auparavant.

On peut penser que c'est inutile de tester si la phrase est infinitive ou non, puisque le traitement est le même. La différence est que, si la phrase source est infinitive et la phrase cible ne l'est pas, la conjugaison de la phrase enchassée dépend du verbe gouverneur. Au contraire, dans le cas où la phrase source est conjugée, la phrase cible sera conjugée selon la source.

3.3 Traitement du mouvement_wh

On a traité quelques mouvements_wh

- direct : who is coming.
- indirect : John wonders who Mary will invite
- relatif : the man who Mary likes
- possessif : the mouse whose eyes Mary likes

3.3.1 Wh_direct

3.3.2 Wh_indirect

On a traité deux types de wh_indirect:

(19)a. John wonders if Mary will come.

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ John] [$_{T}$, wonders [$_{CP}$ if [$_{TP}$ [$_{DP}$ Mary] [$_{T}$, will [$_{VP}$ come]]]]]]]

(20)a. John wonders who Mary will invite.

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ John] [$_{T}$, wonders [$_{CP}$ [$_{DP}$ who]$_i$ [$_{C}$, [$_{TP}$ [$_{DP}$ Mary] [$_{T}$, will [$_{VP}$ invite [$_{DP}$ e]$_i$]]]]]]]]

Puisqu'on a déjà parlé du premier cas concernant les constructions infinitives, on n'y revient pas. Le deuxième cas est presque identique, excepté que la tête du CP est NIL. Donc, on ne crée pas de nœud "si". Mais, lorsqu'on a traité les arguments de la phrase enchassée, on a dû transférer une trace qui représentait l'élément_wh "who". Cet élément n'a pas été mis directement dans la table d'arguments. C'est sa trace qu'on a mis dans la table d'arguments. Le nœud est mis dans la variable **t_ArgWH** qui le projettera dans le spécificateur du complémenteur (CP) au moyen de la procédure **TranslateCP**.

3.3.3 Wh_relatif

Pour cette construction, il suffit de savoir si l'élément_wh, en occurance "who", est le sujet ou l'objet_direct. Dans le premier cas il sera transféré comme "qui" dans **TrEmptyNode** et, dans le second, comme "que". Une fois que l'argument est transféré, on crée sa trace, que l'on met dans la table d'arguments. On met le nœud dans la variable **t_ArgWH** qui le projettera dans le spécificateur du complémenteur (CP) au moyen de la procédure **TranslateCP**.

3.3.4 Wh_possessif

Cette construction est différentes pour les deux traductions.

Anglais/français On reconnait une construction wh_possessive lorsqu'on traite le complément du nom (NP). Si le complément est un CP, alors on a une construction wh_possessive.

(21)a. the cat whose eyes Mary likes.

b. $[_{DP}$ the$[_{NP}$ cat$[_{CP}$ $[_{DP}$ $[_{DP}$ whose$][_{D'}$ $[_{NP}$ eyes$]]]_i[_{C'}$ $[_{TP}$ $[_{DP}$ Mary$][_{T'}$ likes$[_{DP}$ e$]_i$ $]]]]]]$

Contrairement aux phrases simples, on commence par la procédure **TranslateHead**, car le premier syntagme est un déterminant (DP). Une fois qu'il a été traité, on transfère son complément **EFTraverseCompl**, qui est le syntagme nominal. Après le transfert de ce dernier, on passe à son complément **TrComplNP**. C'est à ce moment qu'on fait le test pour déterminer le type de construction. Si le complément est un CP, alors on sauvegarde les nœuds transférés jusque là dans le pointeur globale infoPtr:t_wh_antecedent. On traite le CP comme une phrase normale. Lorsqu'on arrive à la trace, on remonte jusqu'au nœud non_vide à l'aide de la procédure **TrEmptyNode**. Comme c'est un DP, c'est la procédure **TranslateHead** qui est invoquée. On doit traiter ce cas comme celui des syntagmes génitifs, car la structure est très semblable. On teste si c'est un déterminant possessif_wh **IsPossessifWh**. Si oui, on fait appel à la procédure **CreatePossessifWh**. Les phrases comme

(22) whose cat is eating the mouse?

possède les mêmes traits que les phrases wh_possessives, sauf que le pointeur infoPtr:t_wh_antecedent est NIL. Donc, on les traite ensemble. L'algorithme est le suivant:

- création d'un déterminant défini
- extraction et transfert du complément

- mise à jour des traits, entre le déterminant et le nom
- test du pointeur infoPtr:t_wh_antecedent qui indique la construction possessive_wh
  - si le pointeur est différent de NIL alors
    * création de la préposition "dont"
    * mettre ce nœud dans le pointeur infoPtr:t_wh_node, qui ensuite lors de la procédure **TranslateCP** va le projeter dans le spécificateur du (CP).
    * création de la trace de la préposition
    * ajouter la trace comme complément du nœud transféré
    * ajouter le nœud transféré comme complément du déterminant
  - sinon
    * création d'un pronom_wh "qui" et d'une préposition "de"
    * ajouter le pronom comme le complément de la préposition
    * ajouter ce dernier comme complément du nœud transféré
    * création de la trace de ce dernier
    * le mettre dans le pointeur infoPtr:t_wh_node, pour la même raison qu'auparavant.

3.4 Les constructions modales

La construction modale est différente pour les deux traductions, car en anglais une phrase modale est mono_propositionnelle contrairement au français où elle est bi_propositionnelle. L'autre différence importante entre l'anglais et le français est qu'en anglais le verbe modal est considéré comme un auxiliaire, contrairement au français où c'est un verbe à part entière qui requient une construction infinitive.

Anglais/français La construction modale consiste à créer une phrase bi_propositionnelle (phrase infinitive) à partir d'une phrase mono_propositionnelle **ModalConst**.

(23)a. John must come

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ John] [$_{T}$, must [$_{VP}$ come]]]]

c. [$_{CP}$ [$_{TP}$ [$_{DP}$ John]$_i$ [$_{T}$, doit [$_{CP}$ [$_{TP}$ [$_{DP}$ e]$_i$ [$_{T}$, [$_{VP}$ venir]]]]]]]

En géneral, on ne transfére jamais le premier verbe (le verbe de temps), sauf lorsqu'on transfère les phrases modales ou les phrases causatives.

On transfère les arguments comme pour les phrases simples: ils sont placés mis dans la table d'arguments du verbe principal. Ensuite, on commence la construction modale. L'algorithme de cette construction est la suivante:

- on transfère le verbe modal (le premier verbe), mais uniquement la tête **VerbTranslate**
- on construit un complémenteur vide (CP)
- le modal devient le verbe gouverneur et le verbe principal avec ses arguments est considéré comme une phrase enchassée

- le sujet du verbe principal devient le sujet du verbe modal
- le verbe principal est mis à l'infinitif, comme pour les constructions infinitives
- on applique la construction infinitive sur le verbe principal avec contrôle de sujet
- on construit la phrase enchassée
- la phrase enchassée devient le S_object du verbe modal (verbe gouverneur)
- on conjuge le verbe modal **TenseTransfer**

On peut avoir des phrases modales passives où le traitement passif est identique au traitement passif des phrases simple **ApplyPassifModal**

3.5 Les constructions pronominales

Dans ce document, les pronoms faibles ou atones (me, te, se etc.) sont appelés des clitiques et les autres (moi, toi, moi-même, etc.) sont appelés des pronoms toniques ou forts. Cette la construction pronominale est très différente dans les deux langues, car en anglais la cliticisation n'existe pas.

Anglais/français Dans la construction pronominale, on s'intéresse aux pronoms suivants:

1. inhérents
2. anaphoriques (réflechis/réciproques)
3. personnels
4. génitif
5. oblique

L'algorithme pour la construction pronominale est basé sur le document Laenzlinger, Notes techniques 93/5. Entre autre, dans ce document, il explique les groupes de clitiques ainsi que l'ordre et les possibilités combinatoires. Il est important de lire ce document avant de lire cette partie.

Dans cette section, on va reprendre les parties importantes pour la compréhension des algorithmes implémentés pour cette construction. On va tout d'abord parler des groupes et ensuite du traitement des différents pronoms dans l'ordre énuméré ci-dessus. Les groupes de clitiques sont dénotés de la manière suivante:

- seCl : me, te, se, nous, vous
- leCl : le, la, les
- luiCl : lui, leur
- yCl : y
- enCl : en

Les pronoms clitiques traités ici occupent une position préverbale. De ce fait, il faut ajouter le trait **proclitiques** dans le verbe principal dès qu'un pronom est cliticisé. Lorsqu'il y a plusieurs clitiques dans une même phrase, il existe un certain ordre d'apparition qui est : seCl, leCl, luiCl, yCl, enCl. Lorsqu'un pronom est cliticisé, il est mis dans un tableau de pointeur **clitics** qui se trouve dans le verbe principal. Dans ce tableau, chaque groupe ne possède qu'une seule position, c'est-à-dire qu'on ne peut avoir qu'un seul clitique par groupe. De ce fait, l'ordre du traitement des pronoms est important. On a expliqué dans l'ordre de traitement. Certaines combinaisons de clitiques sont impossibles. Ces combinaisons sont décrites dans la procédure booléan **IsCliticPossible**, qui est la suivante:

- deux clitiques du même groupe sont impossibles
- seCl-luiCl est possible seulement s'il existe déjà un clitique du groupe leCl
- luiCl-yCl est impossible
- yCl-leCl est possible que si le clitique du groupe leCl est pluriel

Les pronoms (toniques ou clitiques) ne sont pas cherchés dans le lexique, mais sont créés et modifiés selon les besoins à l'aide des procédures ad-hoc (il faudrait éventuellement les changer plus tard).

Une autre chose à noter est que les pronoms clitiques sont toujours placés devant le premier verbe. Donc, si la phrase a un temps composé comme le passé composé, le verbe principal n'est pas le premier verbe de la phrase. Une fois qu'on a appliqué les transformations de temps, on met tous les pronoms clitiques sur le premier verbe. Ceci a lieu vers la fin de la procédure **TranslateVerbNode**.

3.5.1 Les pronoms inhérents

On commence par traiter les pronoms inhérents. Ils n'ont pas de fonctions grammaticales ou sémantiques, mais sont créés lorsque le verbe est pronominal. Ceci a lieu dans la procédure **Trarguments** lors du transfert des arguments. Une fois que le sujet a été transféré, on teste si le verbe est pronominal **IsNP_NP_Verb**. Si oui, on crée le pronom clitique cible **se** et on modifie ses traits **UpdateHeadFeat** (personne, genre et nombre) selon le sujet ainsi que la chaîne de caractères **UpdateClitic**. Ensuite, on le met dans la table de clitique du verbe principal et, bien sûr, on ajoute le trait préverbal.

3.5.2 Les pronoms anaphoriques

Les pronoms anaphoriques se cliticisent par absorption, c'est-à-dire qu'ils ne laissent pas de trace mais on modifie au niveau du lexème du verbe principal. L'algorithme **Absorption** est décrit plus tard. Le traitement des pronoms anaphoriques a lieu lors du transfert des arguments **Trarguments**, à savoir lorsqu'on traite l'objet_direct et le prep_objet. Le traitement diffère peu pour ces deux arguments. Si l'argument est un prep_objet et si c'est un pronom anaphorique, il ne peut se cliticiser que s'il n'y a pas d'objet_direct ou, s'il y en a, il faut que ce soit un pronom personnel autre que 1,2,4,5 **IsReflexive**. Le reste est identique. On peut décrire l'algorithme de la manière suivante:

- objet_direct et c'est un pronom anaphorique, alors

  - si l'absorption est possible **AbsorptionPossible**
    * on cliticise l'argument **Reflexive**
    * on enlève cet élément de la table d'argument **t_mapTable** et on ne le met pas comme argument dans le verbe principal car il est absorbé
  - sinon, le pronom anaphorique aura la forme tonique **PronounTonic**

- prep_objet et c'est un pronom anaphorique et la préposition est de type "à", alors
  - si l'absorption est possible **AbsorptionPossible** ainsi que la cliticisation **IsReflexive**
    * on cliticise l'argument **Reflexive**
    * on enlève cet élément de la table d'argument **t_mapTable** et on ne le met pas comme argument dans le verbe principal car il est absorbé
  - sinon, le pronom anaphorique aura la forme tonique **PronounTonic**

Comme on l'a déjà dit, la cliticisation des pronoms anaphoriques se fait par absorption de l'argument. Pour qu'un argument puisse être absorbé, il faut qu'il satisfasse les conditions suivantes **AbsorptionPossible**:

- si la phrase est infinitive, il ne faut pas qu'elle soit une construction à montée "raising verb"
- la phrase ne doit pas être passive
- le verbe ne doit pas être conjugé avec l'auxiliaire "être", ceci pour éviter d'avoir plus d'un argument absorbé
- il faut que le clitique soit une combinaison possible **IsCliticPossible**

Pour absorber l'argument **Absorption**, il faut modifier le lexème du verbe principal. Avant d'absorber, il faut tester le statut de cet argument et il faut qu'il soit un objet_direct ou un prep_objet dont la préposition est de type "à". Si l'argument satisfait ces conditions, son statut devient "absorbed" et on ajoute l'auxiliaire "être". Une fois que l'argument est absorbé, on cliticise **Reflexive**.

- on absorbe l'argument
- on crée le pronom clitique **se**
- on modifie ses traits **UpdateHeadFeat** ainsi que la chaîne de caractères **UpdateClitic** selon l'argument source
- on met dans la table de clitiques du verbe principal **CliticTable**
- on ajoute le trait **proclitiques** dans le verbe principal

3.5.3 Les pronoms personnels

Les pronoms personnels se cliticisent en laissant des traces, soit des traces DP si c'est un objet_direct, soit des traces PP si c'est un prep_objet. Le traitement commence aussi dans le **Trarguments** lorsqu'on transfère l'objet_direct et le prep_objet. Mais le traitement proprement dit de cliticisation se trouve dans la procédure **CreateClitics** qui est dans **ApplyTransf**. La raison en est que la cliticisation des pronoms personnels doit être appliquée après toutes les transformations comme le passif, la restructuration, etc.

Lorsqu'on transfère l'objet_direct ou le prep_objet on teste si c'est un pronom personnel. Si c'est le cas, on crée un pronom personnel cible (français) "eux" et on modifie ses traits (personne, genre et nombre sans modifier la chaîne de caractères) selon l'argument source. Une fois que la cliticisation a eu lieu, on trouvera la forme adéquate du pronom (soit la forme clitique soit la forme tonique).

L'algorithme de la procédure CreateClitics est le suivant:

- on parcourt le verbe principal en cherchant les objet_directs et les prep_objets qui sont des pronoms personnels
  - objet_direct
    - on choisit son groupe de clitiques[3] **ChooseClitic**
    - on teste la possiblité combinatoire du clitique **IsCliticPossible**
    - si oui alors **ProcessClitics**
      - on crée le pronom clitique cible "le" et on modifie ses traits selon le pronom personnel "eux" ainsi que la chaîne de caractères
      - on crée la trace du pronoms clitique **CreateTrace**. La trace est de même catégorie que le pronom personnel "eux", donc de catégorie DP
      - cette trace va être mise dans la table d'arguments du verbe principal
      - le pronom clitique est mis dans la table de clitiques **CliticTable**
      - on ajoute le trait **proclitiques** dans le verbe principal
    - sinon, le pronom personnel aura la forme tonique **PronounTonic**, sans trace bien sûr. On doit uniquement changer la chaîne de caractères, car les traits sont déjà modifiés.
  - prep_objets
    - on teste la préposition du prép_object
      - avec la préposition "**de**", la cliticisation est possible uniquement si le pronom personnel est de personnes {3,6} et dans ce cas là le clitique est du groupe **enCl**. La suite est le même que dans le cas d'objet_direct
      - avec la préposition "**à**" le traitement est identique au cas d'objet_direct, sauf que la trace est de catégorie PP
      - pour les autres prépositions la cliticisation est impossible; donc le pronom personnel aura la forme tonique.

3.5.4 Les pronoms génitifs

Le traitement de ce genre de pronom est un peu plus complexe, car il peut être à plusieurs niveaux, comme :

[3]le groupe dépend de la fonction grammaticale, personne et genre; cf document

- argument (déjà mentionné plus haut, avec la préposition "de")

  John talks of it — John en parle

- complément du déterminent

  John eats some of it – John en mange

- complément du nom

  John knows the reason of it – John en connaît la raison

Le premier cas étant déjà expliqué, on n'y revient pas. Pour les deux autres cas, il est impératif que la projection du déterminant ou le syntagme nominal soit des objet_directs. Les pronoms ne sont pas visibles au niveau du **Trarguments**, mais plus antérieurement. Le problème est que, lorsqu'on descend dans l'arbre, on n'a plus accès au verbe principal. Donc, lorsqu'on traite l'objet_direct, on initialise, le pointeur **infoPtr:clitic** sur le verbe principal. En traitant le complément de l'objet_direct, on teste lorsqu'on a un

- déterminant s'il est quantitatif ou numéral ou indéfini dans la procédure **EFTraverseCompl** et si le complément du complément est un pronom personnel de personne {3,6} **IsCliticEN**. Si oui, on cliticise comme dans le cas précédent **CreateCliticEN** (avec trace).

- syntagme nominal si le complément est une préposition **TrComplNP**. Si oui, on teste comme dans le cas du déterminant.

3.5.5 Les pronoms obliques

Les pronoms obliques sont traités comme les ajouts locatifs, et non comme des arguments. Lorsqu'on parcourt le complément du verbe, on teste dans la procédure **TranslateHead** si c'est un pronom de location. Si c'est le cas, on cliticise avec trace.

3.6 Les constructions causatives

Les deux traductions ont des approches différentes pour la construction causative. La raison est que les phrases causatives anglaises sont bi_propositionnelles, alorsqu'en français ce sont des phrases mono_propositionnelles dont le verbe principal a été restructuré. Le traitement des causatives est basé sur le document Wehrli, Notes techniques 93/6.

Anglais/français Le traitement des phrases causatives demande un traitement complètement différent par rapport aux phrases simples. Comme pour les phrases simples on arrive à la procédure **TranslateVerbNode** où l'on teste si le verbe source possède le trait causatif. Toutefois, ceci n'est pas suffisant car certains verbes causatifs ne sont pas traduits comme des causatifs, mais formeront plutôt des phrases infinitives avec contrôle d'objet. Donc, il faut aussi tester si le verbe cible possède le trait causatif. Si oui, on fait appel à la procédure **TrCausative**

Il existe deux types de construction possibles pour les phrases causatives. La première construction est appelée **faire_à** et la seconde est appelée **faire_par**. Le choix entre ces deux structures est assez simple.

(24)a. John makes Mary eat

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ John] [$_{T}$, makes [$_{CP}$ [$_{TP}$ [$_{DP}$ Mary] [$_{T}$, [$_{VP}$ eat]]]]]]]

(25)a. John had the car fixed

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ John] [$_{T}$, had [$_{FP}$ [$_{DP}$ John [$_{NP}$ car]]$_{i}$ [$_{F}$, [$_{VP}$ fixed [$_{DP}$ e]i]]]]]]

Dans le premier exemple le complément du verbe principal (makes) est un S_object (CP). Par contre, dans le deuxième exemple, c'est un Pred_object (FP). Donc, si le complément est un S_object, alors la construction sera **faire_à**, sinon, ce sera **faire_par**.

La différence fondamentale entre les phrases causatives et les autres est qu' on doit faire appel à un processus de restructuration dans le premier cas, c'est-à-dire qu'on doit combiner le verbe causatif (ou verbe opérateur) avec un verbe infinitif. En d'autres termes, on va construire un nouveau lexème qui est une extension du verbe infinitif et qui contiendra les arguments des deux verbes. Cette combinaison va modifier bien sûr l'ordre des arguments, dans le sens que le sujet du verbe infinitif, qu'on appelle le sujet logique, va être mis à la fin des arguments, à condition qu'il n'y ait pas de S_object dans le verbe infinitif. S'il y en a un, alors le sujet logique va être positionné juste avant le S_object. La place libérée par le sujet logique va être occupée par le sujet du verbe opérateur. Lors du déplacement du sujet logique, sa fonction grammaticale va changer, et elle sera déterminée lors du processus de restructuration.

Le processus de restructuration **RestructureVerb** est appliqué sur le verbe infinitif. Donc, il faut d'abord le chercher **GetMainVerb** et ensuite le traduire **TranslateVerbHead**. Comme ce processus modifie le lexème, il faut déterminer si la phrase causative est une construction **faire_à** ou **faire_par**. Le trait **vf3** indique que c'est une **faire_par**. Ce trait est mis dans le verbe cible infinitif.

Au cours de ce processus, on commence par reconnaître si le verbe infinitif est intransitif ou transitif. Ce test est simple: il suffit de trouver la prochaine fonction grammaticale libre **NextArgSlot**. Si la fonction est un objet_direct (c'est-à-dire qu'il n'a pas d'objet_direct), alors le verbe est intransitif. Si elle est un prep_objet (il y a déjà un objet_direct), alors le verbe est transitif. Ceci nous donne la nouvelle fonction grammaticale du sujet logique. Si la fonction grammaticale est:

- objet_direct (* intransitif *)
  - on décale les arguments vers la droite **ShiftArguments**
  - on décale les arguments de la table d'arguments **t_mapTable**
  - on incrémente le nombre d'arguments
  - on marque le sujet logique par le trait **theta6** dans la structure argumentale
- prep_objet (* transitif *)
  - on incrémente le nombre d'arguments
  - on met le sujet logique à la fin de la table d'arguments **t_mapTable**
  - on met le sujet logique à la fin de la structure argumentale en le marquant du trait **theta6**

- on teste le type de construction. Si c'est faire_par, alors on doit mettre la préposition "par". Sinon on introduit la préposition "à".

Une fois que le verbe est restructuré :

- on ajoute dans la table d'arguments **t_mapTable** à la première position le sujet du verbe opérateur
- on transfère les arguments **Trarguments** comme dans le cas des phrases simples
- on applique la cliticisation **CreateClitics** (cf ci-dessous)
- on applique le transfert du temps **TenseTransfer**
- on construit l'arbre cible
- on parcourt le spécificateur et le complément du verbe opérateur
- le verbe principal source devient le verbe infinitif et on continue comme pour les phrases simples

3.6.1 La cliticisation dans les causatives

La cliticisation dans les causatives est un peu plus compliquée que dans le cas des phrases simples. Une description du processus est donnée dans le document précité. Il y a trois cas de clitiques à traiter:

- si on a une construction **faire_par**, on cliticise sans restriction, c'est-à-dire comme pour les phrases simples: les clitiques vont être attachés sur le verbe le plus haut ("faire" ou auxiliaire).
- si on a une construction **faire_à** il est important de savoir si le verbe infinitif est soit pronominal ou s'il est accompagnée d'un pronom anaphorique. Dans chacun des cas, le verbe subit une restructuration faible: tous les clitiques se placent sur le verbe infinitif, sauf le sujet logique, lequel se place sur le verbe le plus haut.
- sinon, il est question d'une restructuration forte à la suite de laquelle tous les clitiques vont être placés sur le verbe le plus haut. Mais, dans certain cas de cliticisation, la construction **faire_à** devient une construction **faire_par**.

Le premier cas ne pose pas de problème. Donc commençons par traiter le dernier cas. La restructuration forte est marquée par le trait r_1. Il y a deux niveaux de restrictions dans cette restructuration.

La première restriction se trouve dans la procédure CreateClitics au moment où l'on cliticise un pronom personnel dont la fonction grammaticale est prep_objet et qui est un clitique est du groupe soit seCl soit luiCl

- si c'est le sujet logique et s'il existe déjà un objet_direct qui est un pronom personnel du groupe leCl, alors on force la construction **faire_par** (il suffit de modifier la préposition) et le pronom est sous forme tonique

- si ce n'est pas le sujet logique et si le clitique est du groupe
  - luiCl, alors on le cliticise à condition qu'il n'y ait pas d'autres clitiques, sinon le pronom est sous forme tonique. Toutefois s'il y a, il faut qu'il y ait un clitique du groupe leCl et un de seCl.
  - seCl, alors on peut le cliticiser à condition qu'il n'y a pas un clitique du groupe leCl, dans quel cas le pronom est sous forme tonique.

La deuxième restriction est appliquée après la procédure ci-dessus. Si l'on a quand même des clitiques du groupe seCl ou luiCl, on vérifie si ce clitique n'est pas le sujet logique. Si c'est le cas, on force la construction **faire_par**. Cette restriction est dans la procédure **FaireAtoPar** et son algorithme est le suivant:

- on cherche le sujet logique
- si c'est un clitique, on ne fait rien
- sinon, on teste la fonction grammaticale du sujet logique
  - si c'est un objet_direct, alors il devient un prep_objet avec la préposition "par"
  - si c'est un prep_objet, la préposition devient "par"

Dans le cas d'une restructuration faible (r_2), il est possible d'avoir des clitiques sur les deux verbes, donc on doit utiliser deux tables de clitiques. Dans le premier tableau **clitic** on met les clitiques du verbe infinitif et dans **cliTable** contient le clitique du sujet logique. Les deux tableaux se trouvent au niveau du verbe.

Avant de faire appel à la procédure **RestructureVerb**, on parcourt la table d'arguments du verbe infinitif et on teste parmi les arguments à part le sujet logique s'il y a des pronoms anaphoriques. Si oui, on absorbe comme dans le cas des phrases simples et on le marque du trait (r_2). Lorsqu'on a un pronom anaphorique, on vérifie si c'est le sujet logique, lors du transfert des arguments. Si oui, on ajoute le trait **vf2** qui nous dira que, lorsqu'on veut faire des vérifications sur les possiblités combinatoires des clitiques ou même les tests d'aborption, on s'intéresse au deuxième tableau. De plus, lorsqu'on cliticise le sujet logique, on le met dans le deuxième tableau. Le reste est identique aux causatives normales. Un autre endroit où il est nécessaire de faire attention est lorsqu'on repositionne les clitiques, donc à la fin de **TranslateVerbNode**. Si le deuxième tableau cliTable est différent de NIL, alors les clitiques doivent monter sur le verbe le plus haut. Si le verbe contient le trait r_2, alors les clitiques du premier tableau clitic ne doivent pas monter, mais doivent plutôt rester sur ce verbe.

3.7 L'accord du participe passé

La notion d'accord du participe passé n'existe qu'en français. Cette partie est basée sur le document Laenzlinger, Notes techniques 93/7. Il est important de le consulter auparavant. L'accord du participe passé **AccParticipe** entervient après le transfert du temps. Cette procédure est activée uniquement si le verbe principal contient le trait **participePasse**. Il y a trois types de verbe où l'accord du participe passé ne doit pas être invoqué. Ces verbes sont les suivants:

- les modaux

- les causatifs
- les impersonnels

Dans la procédure **AccParticipe**, on a traité les cas suivant:

- les constituants_wh
  - (26) La fille qu'il a épousée est belle.
    L'objet_direct est avant le verbe, et comme c'est l'élément_wh "la fille", il est contenu dans le pointeur globale infoPtr:t_wh_antecedent.
  - (27) La fille qui est venue.
    Dans ce cas il n'y a pas d'objet_direct, mais le verbe utilise l'auxiliaire "être" d'où l'accord avec le sujet. Le sujet est contenu dans le même pointeur mentionné ci-dessus, car c'est l'élément_wh de la phrase relative.
  - (28) La souris de qui est-ce que Mary a regardée
    L'objet_direct est avant le verbe, mais ici l'élément_wh est contenu dans un autre pointeur infoPtr:t_wh_node (cf. le traitement des wh_possessifs).
  - Dans les autres cas de wh, il n'y a pas d'accord.
- les phrases dont l'objet_direct est un clitique (avec trace uniquement) autre que du groupe enCl, il y a accord, car le clitique est toujours avant le verbe
- les verbes séléctionnant l'auxiliaire "être" lexicalement s'accordent avec le sujet. Dans les phrases passives, le participe s'accorde avec l'objet_direct (l'objet_direct se trouve à la position du sujet)
- les phrases avec pronom anaphorique (absorbé)
  - lorsque le verbe n'est pas pronominal et qu'il y a un pronom absorbé, il y a accord si et seulement si le pronom absorbé est l'objet_direct et s'il n'y a pas déjà eu un accord du participe passé
  - lorsque le verbe est pronominal, il y a accord si et seulement s'il y n'a pas d'objet_direct ni de S_object
- il n'y a pas d'accord dans les autres cas

On peut remarquer que très souvent on fait appel à **ChooseGender**. La raison en est que les pronoms ont parfois les deux traits masculin, feminin. Cette ambiguïté nous pose un problème lorsqu'on doit cherche le bon participe passé dans le lexique **GetVerbForm**. Si on ne choisit pas le mode intéractif, on prend le masculin par défaut.

Une autre chose à noter concerne le traitement des phrases comparatives au passé. Dans le lexique, le seul participe passé qui existe pour le verbe "être" est "été" avec genre masculin, feminin et personnes {3,6}. Donc, il est important de laisser les deux genres pour que la procédure **GetVerbForm** puisse le trouver.

3.8 Les prédicats adjectivaux

3.8.1 Les constructions tough

Cette transformation a lieu lorsqu'on a un adjectif marqué du trait **tough** comme (easy, difficult, etc) et il a comme complément un infinitif. Il est important que le verbe infinitif ait un objet_direct sinon dans le cas d'un prep_objet le tough doit être bloqué. L'idée de cette transformation est de projetter l'objet_direct de la phrase infinitive à la position sujet de la phrase principale.

(29)a. the book is easy to read

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ the[$_{NP}$ book]]$_i$[$_{T}$, is[$_{FP}$ [$_{DP}$ e]$_i$[$_{F}$, [$_{AP}$ [$_{DP}$ e]$_i$[$_{A}$, easy[$_{CP}$ [$_{DP}$ e]$_i$[$_{C}$, [$_{TP}$ [$_{DP}$ e][$_{T}$, to[$_{VP}$ read catiDPei]]]]]]]]]]]

c. [$_{CP}$ [$_{TP}$ [$_{DP}$ le[$_{NP}$ livre]]$_i$[$_{T}$, est[$_{FP}$ [$_{DP}$ e]$_i$[$_{F}$, [$_{AP}$ [$_{DP}$ e]$_i$[$_{A}$, facile[$_{CP}$ [$_{DP}$ e]$_i$[$_{C}$, à[$_{TP}$ [$_{DP}$ e][$_{T}$, [$_{VP}$ lire catiDPei]]]]]]]]]]]

Lorsqu'on bloque ce mouvement la traduction a la forme suivante:

(30)a. Paul is difficult to talk to

b. [$_{CP}$ [$_{TP}$ [$_{DP}$ Paul]$_i$[$_{T}$, is[$_{FP}$ [$_{DP}$ e]$_i$ [$_{F}$, [$_{AP}$ [$_{DP}$ e]$_i$[$_{A}$, difficult[$_{CP}$ [$_{DP}$ e]$_i$ [$_{C}$, [$_{TP}$ [$_{DP}$ e][$_{T}$, to[$_{VP}$ talk[$_{PP}$ to[$_{DP}$ e]$_i$]]]]]]]]]]]]

c. [$_{CP}$ [$_{TP}$ [$_{DP}$ il]$_i$[$_{T}$, est[$_{FP}$ [$_{DP}$ e]i [$_{F}$, [$_{AP}$ [$_{DP}$ e]$_i$[$_{A}$, difficult[$_{CP}$ de[$_{TP}$ [$_{DP}$ e][$_{T}$, [$_{VP}$ parler[$_{PP}$ à[$_{DP}$ Paul]]]]]]]]]]]]

On va bien que lorsqu'on bloque ce mouvement le prep_objet n'est pas projeté vers le haut et deplus on doit créer un nœud "il".

L'algorithme est le même que lorsqu'on traite les prédicats adjectivaux, mais au moment du traitement du complément de l'adjectif **TrComplAP** et si ce dernier est un CP alors on savegarde dans le pointeur global **infoPtr:govern_adj** l'adjectif cible ainsi que son sujet car à ce niveau on ne sait pas encors si le mouvement est applicable ou non. On le saura que lorsqu'on traite les arguements de ce CP. Le traitement de ce CP est le même que pour les autres CP. Lorsqu'on arrive à la procédure Trarguments on transfére les arguments normalement et ensuite on fait les tests suivant selon la fonction grammaticale, si l'argument est un

- objet_direct on test si ce dernier et le sujet de l'adjectif (le spécificateur de l'élément stocké dans le pointeur) sont identiques si oui on crée la trace du sujet, et on le met dans l'objet_direct (c'est-à-dire l'objet_direct devient une trace). Ceci est traité dans la procédure **ToughMovement**
- prep_odjet on doit bloquer ce mouvement, donc on fait appel à la procédure **Block-Tough**. Dans cette procédure on enlève le sujet de l'adjectif. C'est cette différence qui nous permet de savoir si le mouvement est bloqué ou non.

A la fin de la procédure **TranslateCP** on teste si on a un adjectif tough si oui on teste si le mouvement doit être bloqué ou non. Si le mouvement n'est pas bloqué alors on doit mettre

dans le spécificateur du CP la trace de l'objet_direct (qui le sujet de l'adjectif) et dans la tête du CP il faut ajouter la préposition "à". Si le mouvement est bloqué alors on doit mettre dans la tête du spécificateur la préposition "de".

Comme pour tous les pred_objets on doit passer par la procédure **RaiseSpecFP**. C'est ici où l'objet_direct monte à la position sujet de la phrase principale en laissant la trace de la montée dans le spécificateur de l'adjectif (AP) et dans le spécificateur de FP, si bien sûr le mouvement n'est pas bloqué.

Si le mouvement est bloqué alors on crée le nœud "il" qui devient le sujet de l'adjectif et va être projeté à la position sujet en laissant des traces dans le spécificateur de AP et dans le spécificateur de FP.