Structures de traits typées et morphologie à partitions

François BARTHÉLEMY[1, 2]
[1] CNAM, Cédric, 292 rue Saint-Martin, 75003 Paris
[2] INRIA, Atoll, 78153 Le Chesnay cedex
barthe@cnam.fr

Résumé. Les structures de traits typées sont une façon abstraite et agréable de représenter une information partielle. Dans cet article, nous montrons comment la combinaison de deux techniques relativement classiques permet de définir une variante de morphologie à deux niveaux intégrant harmonieusement des structures de traits et se compilant en une machine finie. La première de ces techniques est la compilation de structure de traits en expressions régulières, la seconde est la morphologie à partition. Nous illustrons au moyen de deux exemples l'expressivité d'un formalisme qui rapproche les grammaires à deux niveaux des grammaires d'unification.

Abstract. Feature Structures are an abstract and convenient way of representing partial information. In this paper, we show that the combination of two relatively classical techniques makes possible the definition of a variant of two-level morphology which integrates harmoniously feature structures and compiles into finite-state machines. The first technique is the compilation of feature structures into regular expressions, the second one is partition-based morphology. Two examples are given, which show that our formalism is close to unification grammars.

Mots-clés : morphologie à deux niveaux, transducteurs finis à états, structure de traits.

Keywords: two-level morphology, finite-state transducers, feature structures.

1 Introduction

La morphologie à états finis est un courant important de la morphologie informatique qui propose des formalismes de règles contextuelles (grammaires à deux niveaux ou règles de réécriture) pour décrire la morphologie des langues. Ces règles dénotent une relation rationnelle reconnue au moyen d'un transducteur fini.

L'utilisation de structures de traits pour la morphologie à états finis est une pratique relativement courante, que ce soit dans la littérature ou dans les systèmes diffusés comme PC-Kimmo (Antworth, 1995), Xerox Finite-State Tools (Beesley & Karttunen, 2003) ou MMORPH (Petitpierre & Russel, 1995). On peut distinguer deux approches : l'une consiste à compiler les traits statiquement dans les machines finies, l'autre consiste à vérifier les contraintes après exécution de la machine finie au moyen d'une procédure d'unification dynamique. Cette dernière option est coûteuse en temps de calcul à effectuer lors de chaque analyse, mais elle permet d'utiliser

toute la puissance de l'unification. Elle est utilisée notamment dans PC-Kimmo version 2 et dans MMORPH.

La compilation de structure de traits en machines finies impose des contraintes spécifiques qui ont été abordées de deux façons différentes : avec ou sans changement du formalisme des machines finies. Dans la première catégorie, nous trouvons Rémi Zajac (Zajac, 1998) qui propose d'utiliser les structures de traits pour le niveau lexical d'un système de morphologie à deux niveau et remplace sur ce niveau la concaténation par l'unification. Jan Amtrup (Amtrup, 2003) propose quant à lui d'utiliser des machines pondérées par une structure de trait utilisée comme un poids. Cette approche est correcte parce que les structures de traits munies de l'union et de l'unification forment un semi-anneau. La limite de ces deux travaux est que la structure de traits est unique et doit s'enrichir de façon monotone, c'est à dire que les calculs successifs ne peuvent que préciser la valeur des traits, jamais la changer.

L'approche qui consiste à compiler les traits en symboles ordinaires dans des machines finies standards est représentée par XFST d'une part et George Kiraz d'autre part (Kiraz, 1997). Dans XFST, il n'y a pas à proprement parler de structure de traits, mais des traits isolés que l'on peut mentionner à tout endroit dans les expressions régulières pour leur appliquer une opération (fixer, unifier ou redéfinir leur valeur). Ces traits ont une portée globale sur toute une chaîne et les opérations d'évaluation sont effectuées dans un parcours gauche-droite des chaînes. Kiraz propose quant à lui de vraies structures de traits à portée locale, ayant pour seul but un filtrage des règles contextuelle en fonction de traits précisés dans le lexique.

Nous proposons une utilisation plus libre et plus systématique de traits compilés sous forme d'expressions régulières ordinaires, avec la possibilité d'avoir des structures à portée soit locale (par exemple ne concernant qu'un morphème), soit globale (concernant toute une forme), soit encore l'utilisation simultanée de différentes structures de traits ayant des portées différentes. Les traits peuvent être précisés et utilisés aussi bien dans le lexique que dans les règles.

Des restrictions sont apportées à la forme que peuvent prendre les structures de traits ainsi que leurs domaines pour permettre une compilation en expression régulière. Par ailleurs, l'implémentation de la notion de portée d'une structure de trait repose sur les concepts et techniques de la morphologie à partition, une approche de la morphologie à états finis dont le principal contributeur est George Kiraz (Kiraz, 2001).

Dans la section suivante, nous allons voir comment compiler des structures de traits en expressions régulières. Nous verrons ensuite comment ces expressions régulières peuvent être intégrées aux autre composantes d'une description morphologique et nous proposerons un formalisme adéquat. Nous illustrerons l'intérêt de ce formalisme au moyen de deux exemples, l'un n'utilisant qu'une structure de traits globale et l'autre utilisant une véritable grammaire de structure de traits.

2 Compilation des structures de traits

Dans cette section, nous abordons la question de la compilation de structures de traits en automates finis. Plus précisément, nous allons nous intéresser à un sous-ensemble des structures de traits dont la compilation est triviale : il s'agit des structures de traits acycliques prenant leurs valeurs dans des ensembles finis petits.

L'intérêt essentiel de ces structures est d'offrir une syntaxe agréable pour représenter une in-

formation partielle, susceptible d'être complétée via des opérations algébriques (unification ou application de règles).

Dans un premier temps, considérons des structures de traits plates, c'est-à-dire sans structures imbriquées. Chaque trait identifié par son nom prend une valeur dans un ensemble fini de valeurs connu. On peut représenter chaque couple nom-valeur par un symbole spécial et une structure par une chaîne obtenue par concaténation des symboles correspondant à ses différents traits. Prenons par exemple les marques de nombre et personne utiles à décrire la conjugaison du français. Le nombre peut prendre les deux valeurs singulier et pluriel, la personne peut prendre les trois valeurs 1, 2 ou 3. Cela conduit à décrire un alphabet avec les cinq symboles <nombre=singulier>, <nombre=pluriel>, <personne=1>, <personne=2> et <personne=3>. Une structure $[nombre = singulier, personne = 1]$ se compile en la chaîne <nombre=singulier><personne=1>.

Pour assurer l'unicité de la représentation d'une structure, on peut imposer un ordre fixe entre symboles d'une structure basé sur le seul nom des traits, par exemple en utilisant l'ordre lexicographique.

Si l'on connaît à l'avance l'ensemble des traits susceptibles de venir enrichir au fil des calculs un structure de traits, on peut représenter une information partielle au moyen d'une expression régulière représentant l'ensemble des traits. Par exemple, la structure $[personne = 3]$ se compile en (<nombre=singulier>|<nombre=pluriel>)<personne=3>. L'intérêt de cette représentation vient de ce que l'unification de structures de ce genre s'implémente par l'intersection des expressions correspondantes. En définissant une classe de caractères <nom=_> comme l'union des caractères <nom=x> représentants les valeurs que peut prendre le trait `nom`, cette expression peut s'écrire de façon équivalente <nombre=_><personne=3>.

L'unification n'est pas la seule opération que l'on peut désirer réaliser avec des structures de traits. Des règles de grammaires peuvent décrire la construction d'un structure à partir d'une ou plusieurs structures, en spécifiant ce qui doit être emprunté à l'une ou à l'autre au moyen de variables. Par exemple la règle suivante décrit l'adjonction d'un suffixe à une base pourvue de la bonne catégorie syntaxique :

$$\begin{bmatrix}\text{cat} & \boxed{1}\\ \text{nombre} & \boxed{2}\end{bmatrix} \rightarrow \begin{bmatrix}\text{cat} & \boxed{3}\end{bmatrix} \begin{bmatrix}\text{cat} & \boxed{1}\\ \text{de\_cat} & \boxed{3}\\ \text{nombre} & \boxed{2}\end{bmatrix}$$

Le trait `cat` décrit la catégorie syntaxique de la base (premier opérande), du résultat de l'adjonction du suffixe, alors que le trait `de_cat` (second opérande) spécifie la catégorie syntaxique de la base pour que la dérivation soit correcte.

Une telle règle peut être implémentée par un transducteur à trois bandes, une pour chaque opérande et une pour le résultat. Ce transducteur, sous certaines conditions[1], peut être obtenu par intersection de transducteurs implémentant chacun une des variables de la règle. Si l'on suppose que les différents rubans sont synchronisés sur les valeurs des traits, cela donne :
$\boxed{1}$: (_ :_ :_)*(_ :P :P)(_ :_ :_)* where P in <cat=_>
$\boxed{2}$: (_ :_ :_)*(_ :C :C)(_ :_ :_)* where C in <nombre=_>
$\boxed{3}$: (_ :_ :_)*(<cat=X> :_ :_)(_ :_ :_)*(_ :<de_cat=X> :_)(_ :_ :_)*(_ :C :C)(_ :_ :_)*
where X in dom(cat)∩dom(de_cat)

[1] L'intersection de transducteurs n'est pas définie pour les transducteurs en général, mais elle l'est pour certaines sous-classes particulières.

Au sein d'une structure de traits, une variable peut être utilisée pour noter le fait que plusieurs traits partagent une même valeur. Une telle structure est compilée en une disjonction de chaînes, chacune d'elle représentant une des valeurs possibles de la variable.

Le technique de compilation que nous venons de voir s'étend facilement aux structures imbriquées acycliques. Il faut simplement remplacer la notion de nom de trait par celle de chemin. Par exemple, la structure suivante : $\begin{bmatrix} \text{cat} & \text{nom} \\ \text{agr} & \begin{bmatrix} \text{genre} & \text{masc} \\ \text{nombre} & \text{pluriel} \end{bmatrix} \end{bmatrix}$ se compile en <agr.genre=masc><agr.nombre=pluriel><cat=nom>. L'intérêt de cette imbrication est de pouvoir représenter au moyen d'une seule variable l'égalité de tous les traits de la sous-structure.

3 Présentation du formalisme

Le formalisme que nous proposons est basé sur la morphologie à partition. L'historique de ce courant se trouve dans (Kiraz, 2001) alors que sa compilation en automate fini est décrite dans (Barthélemy, 2005). L'idée centrale consiste à définir des relations n-aires dont les différentes chaînes sont divisées en un nombre égal de sous-chaînes. Par exemple, on peut relier une représentation écrite et une représentation phonologique de la façon suivante :

e	x	em	p	l	es
e	gs	ã	p	l	

Comme on le voit, les sous-chaînes mises en correspondances peuvent être de longueurs différentes et éventuellement nulles.

Les relations régulières partitionnées sont la classe de relations qu'on peut décrire avec des expressions régulières augmentées d'une construction nouvelle que nous appellerons tuple, permettant de mettre en relation deux ou plusieurs sous-chaînes. Par exemple, l'expression régulière : $< [lettre]^*, [phoneme]^* >^* < e, \epsilon >$ dénote l'ensemble des chaînes terminées par un e muet. Les opérations comme la concaténation, la disjonction, l'étoile, peuvent intervenir aussi bien à l'intérieur d'un tuple que sur un tuple.

Le formalisme que nous proposons autorise la description de relations n-aires et non seulement binaires, ce qui correspond à une morphologie à n niveaux, n pouvant être différent de 2. Les niveaux supplémentaires peuvent être utilisés soit pour distinguer des facteurs indépendants à un niveau donné, comme c'est le cas par exemple pour la description du Syriaque dans (Kiraz, 2000), soit pour distinguer des niveaux intermédiaires dans une cascade de traitements comme c'est le cas dans l'analyseur morphologique de l'akkadien décrit dans (Barthélemy, 2006).

Le formalisme est fondé sur des expressions régulières étendues pour prendre en compte les notions de partition et de structures de traits. Les règles contextuelles sont admises en tant que raccourcis syntaxiques dénotant des expressions régulières.

Une description comporte les sections suivantes : domaines des traits, types de structures de traits, définition de l'alphabet, types des différents niveaux, types des tuples, autres types d'expressions régulières, définition des machines finies.

Nous allons donner en exemple une description schématique de la conjugaison des verbes français. Nous discuterons cet exemple dans la section suivante. Les points de suspension matérialisent des coupures que nous avons réalisé dans l'exemple pour gagner de la place.

```
PACKAGE verbes;
FEATURES VALUES
   temps: present, futur, passe, imparfait;
   mode: indicatif, subjonctif, conditionnel, imperatif;
   personne: 1, 2, 3;
   nombre: singulier, pluriel;
   conjugaison: 1, 2, 3, irreg;
END VALUES
FEATURE STRUCTURES
   verbe: temps, mode, personne, nombre, conjugaison;
END STRUCTURES
ALPHABET
   [lettre]: a, b, c, d ...
   [voyelle]: a, à, â, e, é ...
   [consonne]: b, c, d ...
END ALPHABET
LEVELS
   1: [verbe:_];
   2: [lettre]+;
   3: [lettre]*;
   4: [lettre]*;
END LEVELS
TUPLES
   <3| LEVEL 3: [lettre], LEVEL 4: [lettre] |3>;
   <2| <3|_|3>* |2>;
   <1| LEVEL 0, LEVEL 1, <2|_|2><2|_|2> |>;
END TUPLES
TYPES
   <radical: LEVEL 1, LEVEL 2, LEVEL 3 > =>
           <1| #1, #2, <2| /LEVEL 3: #3/ |2><2|_|2> |1>;
   <suffixe: LEVEL 2, LEVEL 3 > =>
           <1| _, #1, <2|_|2> <2| /LEVEL 3: #2/ |2>;
END
```

Les domaines de traits sont des listes de valeurs que peuvent prendre les différents traits. D'autres domaines finis de valeurs peuvent également être défini et une même valeur peut appartenir à plusieurs domaines. Les structures de traits sont typées au moyen d'un nom de type associé à la liste des traits de la structure. Dans la syntaxe, le nom de type apparaît en début de structure, suivi de deux points.

Chaque niveau est caractérisé par un numéro et son type est une expression régulière définissant un sur-ensemble des chaînes susceptibles d'être lues sur ce niveau. Le type d'un tuple est constitué de la liste ordonnée de ses niveaux, avec pour chacun d'entre eux une expression régulière restreignant la sous-chaîne pouvant apparaître sur ce niveau. Les différents tuples peuvent différer par leur arité, les niveaux qu'ils comportent et leur degré d'imbrication. D'autres types d'expressions régulières peuvent être définis pour faciliter l'écriture des expressions régulières, par exemple en spécifiant un contenu sous-entendu pour certains niveaux de certains tuples.

Les relations régulières sont nommées. Elles peuvent être définies de trois manières différentes. La première forme est celle d'une expression régulière utilisant des symboles de l'alphabet, des constructeurs de tuples et différentes facilités syntaxiques. Par exemple, on peut utiliser des variables prenant leur valeur dans un ensemble fini. L'expression dénotée est l'union des expressions obtenue par substitution de la variable par une de ses valeurs. Par ailleurs, on autorise l'utilisation d'un joker (wildcard) noté _ dans différents contextes. La projection notée `/LEVEL x: _/` permet de ne spécifier que le contenu d'un niveau dans une expression qui en comporte plusieurs. La construction `REGEXP` réalise implicitement l'union des expressions régulières qu'elle contient, chacune étant terminée par un point-virgule.

```
REGEXP les_radicaux IS
   <radical: lancer, [verbe:conjugaison=1], lanC >;
   <radical: polir, [verbe:conjugaison=2], poli >;
   <radical: pouvoir, [verbe:conjugaison=3], p[OU_EU]v >;
   ...
END
REGEXP terminaisons IS
   <suffixe: [verbe:temps=present,mode=indicatif,
         nombre=singulier,personne=1|3,conjugaison=1], e>;
   <suffixe: [verbe:temps=present,mode=indicatif,
         nombre=singulier,personne=1|2,conjugaison=2|3], s>;
...
END
LET formes=intersect(les_radicaux,terminaisons);
```

Le deuxième moyen de spécifier une relation régulière est par application d'opérations sur des relations définies auparavant. Les opérations comprennent les opérations ensemblistes (union, intersection, différence) et les opérations rationnelles (concaténation, étoile). La projection permet d'éliminer certains niveaux. Sous certaines conditions, l'opération de jointure permet de composer deux relations ayant des domaines différents.

Le dernier moyen de décrire une relation régulière est l'utilisation de règles contextuelles. Ce sont des adaptations aux relations n-aires des règles classiques de la morphologie à deux niveau. Les règles de coercition spécifient un certain motif et contraignent les valeurs que peuvent prendre, en contexte, les sous-chaînes filtrées par ce motif. Par exemple, la règle suivante décrit la réalisation d'un méta-caractère C susceptible de s'écrire ç ou c selon le contexte (comme par exemple dans le verbe lancer, je lançais) :

```
 <3| C, _ |3> => <3| C, ç |3>
    IF _ <2| <3_|3>* XXX |2><2| <3|_, [lettre]-(i|e)|3> _ |2> _
```

Le motif apparaît à gauche de la flèche et la restriction à droite de la flèche. Le contexte est décrit en utilisant `XXX` pour désigner le centre de la règle, à distinguer de _, utilisé ici comme joker. Dans notre système multi-niveaux, il n'y a pas de distinction explicite entre niveau lexical et niveau de surface. N'importe quel ensemble de niveaux peut être précisé dans le motif et deux règles différentes peuvent utiliser des ensembles de niveaux différents, ce qui introduit plus de souplesse et justifie le changement de nom de coercition de surface en coercition tout court.

Une règle de restriction de contexte décrit un contexte dans lequel un motif peut exclusivement apparaître (syntaxe : `motif ONLY IF contexte`). Une règle composite est un règle qui cumule les deux contraintes de coercition et de restriction de contexte.

L'utilisation de règles contextuelles posent des problèmes de conflits, quand deux règles sont d'une certaine façon contradictoires. Nous ne traiterons pas de ce problème en détail dans cet article dont ce n'est pas l'objet. L'existence de ces conflits justifie qu'on considère les règles comme un ensemble et non séparément. La détection des conflits peut être automatisée (Beesley & Karttunen, 2003) et leur résolution peut être aidée par une procédure interactive.

4 Exemple avec structure de traits unique

Dans ce premier exemple, nous voulons insister sur la question du niveau abstrait d'une représentation et promouvoir l'idée que la multiplicité des niveaux permet d'offrir une réponse adéquate. Ce qu'on appelle le niveau lexical dans un système de morphologie à deux niveaux traditionnel est une représentation relativement concrète sur laquelle il faut appliquer quelques transformations pour obtenir une représentation de surface. Il s'agit en fait d'une approximation aussi précise qu'on peut faire de la représentation de surface d'un morphème avant application des mécanismes de dérivation et/ou de flexion.

Dans un système comme PC-Kimmo, la représentation abstraite de la forme est ce qu'on appelle la glose (gloss), une chaîne de caractère précisée dans le lexique et destinée à être affichée en réponse à certaines requêtes. Nous proposons d'inclure cette information dans un ou plusieurs niveaux, sans exclure les deux représentations classiques : approximation avant composition et forme de surface.

Rémi Zajac (Zajac, 1998) propose d'utiliser une structure de traits comme niveau abstrait d'un système à deux niveaux. Nous allons affiner cette idée pour permettre une compilation en machine finie : il faut représenter sous forme d'une structure de trait uniquement les traits élémentaires prenant leur valeur dans un ensemble fini et petit et sous la forme d'une chaîne utilisant un niveau spécifique les informations structurées ou ayant un grand nombre de valeurs.

Dans l'exemple de la conjugaison du verbe français, une forme abstraite doit préciser le lemme, le temps, le mode, la personne, le nombre. Le nombre de lemmes est grand. Créer un symbole par lemme conduit à multiplier le nombre de symboles au-delà de ce qui est couramment accepté par les implémentations de machines finies. Le lemme sera donc noté sous la forme d'une chaîne de caractère et cela constitue le niveau 1. Les autres informations ont peu de valeurs, on les regroupe donc dans une structure de traits qui occupe le niveau 2. Les niveaux 3 et 4 sont consacrés aux représentations intermédiaire et de surface. Par ailleurs, pour coordonner les lemmes et les terminaisons, il faut connaître le paradigme de conjugaison utilisé. Cette information pourrait être mise sur un niveau de service, mais pour simplifier la description, nous la plaçons dans la structure de traits du niveau 1.

Cet exemple illustre comment le typage permet de ne préciser que l'information pertinente pour les radicaux et les terminaisons, tout en ayant une représentation sous-jacente unique, ce qui permet d'opérer une intersection. Cette intersection réalise l'unification des structures de traits spécifiées dans les deux expressions régulières, et notamment l'identification de leur unique trait commun, `conjugaison`. Par exemple,
`<radical: lancer, [verbe:conjugaison=1], lanC >` est une notation équivalente à l'expression :

```
<1| lancer, [verbe:conjugaison=1],
      <2| <3|l,_|3><3|a,_|3><3|n,_|3><3|C,_|3> |2><2|_|2> |1>.
```

Il convient ensuite de compléter la description en précisant comment relier le niveau intermédiaire (niveau 3) avec la réalisation de surface (niveau 4).

```
RULE SET
  <3| $L,$L |3> where $L in [lettre];
  <3| C, c |3>
     ONLY IF _ XXX /Level 3: (e|i) _/;
  <3| s,_ |3> => <3| s,x |3>
     IF <1| [verbe:temps=present,nombre=singulier,
             mode=indicatif],
             pouvoir|vouloir, <2|_|2><2| XXX |2> |1>;
  ...
```

La dernière règle illustre comment une règle contextuelle peut être conditionnée par la valeur des traits en utilisant simplement la notion de contexte habituelle.

5 Exemple avec plusieurs structures de traits

Nous allons prendre comme exemple une grammaire ayant une structure linéaire, décrivant une morphologie basée exclusivement sur des suffixes. Les machines finies permettent de représenter plus facilement de telles structures que des arbres quelconques. Une morphologie basée à la fois sur des préfixes et des suffixes, voire des circonfixes, est plus difficile à traiter. Ces problèmes techniques ne sont pas insurmontables, mais alourdiraient trop notre exemple. Nous allons donc nous limiter à des suffixes susceptibles de changer la catégorie syntaxique d'un mot et donc son type de flexion.

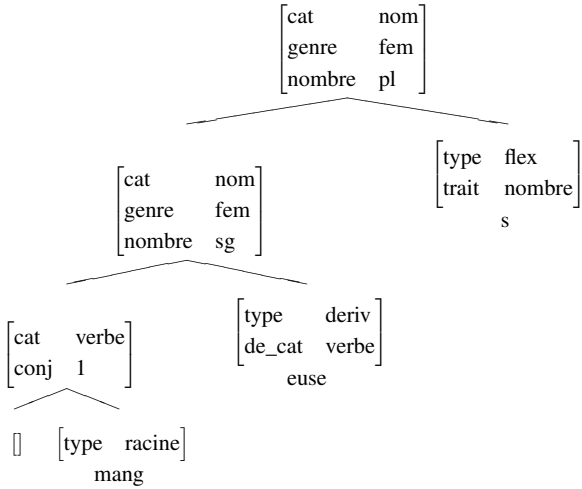

FIG. 1 – Exemple de structure

La figure 1 donne un exemple de structure que nous voulons représenter, celle qui correspond à la forme `mangeuses`. Dans cette structure binaire, des informations doivent être propagées comme par exemple le genre, qui n'est pas modifié par la marque de pluriel. Les suffixes de flexion modifient une partie de la structure, préservant le reste, alors que les suffixes de dérivation bloquent la transmission d'information qui n'est plus pertinente compte tenu du changement de catégorie syntaxique. C'est ici le cas du paradigme de conjugaison.

Nous voyons dans cet exemple que le nombre de noeuds internes de la structure est égal au nombre de morphèmes, ce qui permet d'utiliser le même tuple pour décrire un morphème et son père. Dans chacun de ces tuples, il y aura donc deux structure de traits : une associée au seul morphème, l'autre concernant la structure regroupant le morphème et tous ceux qui le précèdent.

```
REGEXP affixes IS
<affixe: [verbe:cat=verbe,conj=1], [racine], mang >;
<affixe: [nom:cat=nom,genre=fem,nombre=sg],
                              [deriv:de_cat=verbe], euse >;
<affixe: [nom:cat=nom,nombre=pl], [flex:trait=nombre], s >;
...
END
RULE SET composition IS
  <affixe: _, [deriv:de_cat=$A], _ >
      ONLY IF _ <affixe: [_:cat=$A], _ , _> XXX _;
  <affixe: [nom:genre=$G], [flex:trait=nombre], _ >
      ONLY IF _ <affixe: [nom:genre=$G],_,_> XXX _;
  ...
```

La dérivation peut être traitée au moyen d'une règle contextuelle unique qui vérifie que la base possède la catégorie syntaxique requise. Pour la flexion, en revanche, il faut une règle pour chaque type de suffixe, car les traits propagés et les traits révisés ne sont pas les mêmes.

Les deux règles règles données en exemple illustrent comment les différentes structures de traits interagissent et notamment comment certains traits en sont unifiés. Elles pourraient aussi bien s'exprimer sous forme de règle de grammaires de traits. Par exemple la seconde :

$$\begin{bmatrix} \text{nom} & \\ \text{genre} & \boxed{1} \\ \text{nombre} & \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \text{nom} & \\ \text{genre} & \boxed{1} \end{bmatrix} \begin{bmatrix} \text{flex} & \\ \text{nombre} & \boxed{2} \\ \text{trait} & \text{nombre} \end{bmatrix}$$

6 Conclusion

Dans cet article, nous montrons comment l'utilisation simultanée de deux techniques préexistantes, à savoir la compilation de structure de traits en chaînes de caractères et la morphologie à partition, offre un pouvoir de description intéressant.

Il n'y a bien sûr aucune augmentation de puissance du formalisme. Il s'agit de facilité d'écriture : les structures de traits sont pratiques parce qu'on ne précise que l'information connue et que l'ordre des traits n'est pas significatif. De plus ce formalisme est familier aux personnes travaillant dans le TAL.

La technique que nous proposons offre un risque d'explosion de la taille des machines. Ce risque est important si l'on multiplie les traits, les valeurs et surtout les unifications entre structures éloignées. Notre expérience montre que ce risque n'est pas rédhibitoire. Nous avons réalisé un prototype qui compile une description syntaxique en un automate fini, en utilisant la boîte à outils FSM (Mohri *et al.*, 2002). Nous avons écrit des grammaires relativement grosses ($\sim$ 50 règles) sans provoquer d'explosion incontrôlée (Barthélemy, 2006).

L'outil de Xerox (xfst) offre une possibilité intéressante pour éviter l'explosion combinatoire : elle consiste à choisir entre un calcul statique ou dynamique pour les valeurs de traits. Dans le cas d'un calcul dynamique, ce ne sont pas les seules valeurs de traits qui sont représentées sous forme de symboles dans les machines, mais les calculs à réaliser sur ces traits lors d'une évaluation de gauche à droite. A priori, il semble possible d'adapter cette technique à notre formalisme.

Notre proposition permet une utilisation plus générale des traits que les travaux antérieurs proposant une compilation en machine finie. Par rapport à (Zajac, 1998), (Amtrup, 2003), l'apport principal est la notion de portée d'une structure qui peut être locale à un tuple, ce qui autorise la multiplicité de structures ayant certains traits communs dont les valeurs sont indépendantes. Les interactions entre structures de traits sont plus riche que dans (Kiraz, 1997). Par rapport aux approches qui proposent une évaluation dynamique des structures de traits, les gains proviennent d'une meilleure intégration avec les calculs d'automates (par exemple, calcul d'intersection) ainsi qu'une plus grande efficacité.

Références

AMTRUP J. W. (2003). Feature structures as weights in finite state morphology. In *FSMNLP*, Budapest, Hongrie.

ANTWORTH E. L. (1995). User's guide to pc-kimmo version 2.

BARTHÉLEMY F. (2005). Partitioning multitape transducers. In *International Workshop on Finite State Methods in Natural Language Processing (FSMNLP)*, Helsinki, Finlande.

BARTHÉLEMY F. (2006). Un analyseur morphologique utilisant la jointure. In *In Traitement Automatique de la Langue Naturelle (TALN'06)*, Leuven, Belgique.

BEESLEY K. R. & KARTTUNEN L. (2003). *Finite State Morphology*. CSLI Publications.

KIRAZ G. A. (1997). Compiling regular formalisms with rule features into finite-state automata. In *ACL*, Madrid, Espagne.

KIRAZ G. A. (2000). Multitiered nonlinear morphology using multitape finite automata : a case study on syriac and arabic. *Computational Linguistics*, **26**(1), 77–105.

KIRAZ G. A. (2001). *Computational Nonlinear Morphology*. Cambridge University Press.

MOHRI M., PEREIRA F. C. N. & RILEY M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, **16**(1), 69–88.

PETITPIERRE D. & RUSSEL G. (1995). Mmorph : the multex morphology program.

ZAJAC R. (1998). Feature structures, unification and finite-state transducers. In *FSMNLP'98*, Ankara, Turquie.