TALN 2008, Avignon, 9–13 juin 2008

Typage, produit cartésien et unités d'analyse pour les modèles à états finis

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

Résumé. Dans cet article, nous présentons un nouveau langage permettant d'écrire des relations rationnelles compilées en automates finis. Les deux caractéristiques innovantes de ce langage sont de pourvoir décrire des relations à plusieurs niveaux, pas nécessairement deux et d'utiliser diverses unités d'analyse pour exprimer les liens entre niveaux. Cela permet d'aligner de façon fine des représentations multiples.

Abstract. In this paper, we present a new language to write rational relations compiled into finite state automata. There are two main novelties in the language. Firstly, the descriptions may have more than two levels. Secondly, various units may be used to express the relationships between the levels. Using these features, it is possible to align finely multiple representations.

Mots-clés : Machine finie à états, morphologie à deux niveau.

Keywords: Finite-state machine, two-level morphology.

1 Introduction

Les machines à états finis, à savoir les automates et transducteurs, ont une place importante dans plusieurs domaines de la linguistique informatique, notamment la description morphologique, le traitement des entités nommées, les approximations de syntaxe, la recherche de motifs.

Ces machines sont spécifiées au moyen d'expressions régulières, de règles contextuelles ainsi que par l'application d'opérateurs sur des machines existantes. Les automates implémentent des langages rationnels et les transducteurs des relations rationnelles. Ce sont deux types d'objets différents, pouvant être convertis l'un dans l'autre au moyen d'opérations. Une relation peut être obtenue en faisant le produit cartésien de deux langages et un langage peut être obtenu en faisant une projection d'une relation sur sa première ou sa seconde composante. Les opérations rationnelles (concaténation, clôture sous concaténation ou étoile, disjonction) sont définies sur les deux types d'objets. En revanche, certaines opérations ensemblistes, notamment l'intersection et la différence, ne sont des opérations internes que pour les langages rationnels. Les relations pour leur part, sont closes sous composition, une opération qui n'a pas de sens sur les langages.

Dans cet article, nous proposons une extension aux relations n-aires, c'est-à-dire pouvant avoir plus de deux composantes ou niveaux, spécialement adaptée pour des descriptions linguistiques reliant des représentations différentes de certains objets. Cette extension est un langage qui s'inscrit dans la lignée de langages comme PC-Kimmo (Antworth, 1995) ou Xfst (Beesley &

Karttunen, 2003). Il autorise aussi bien les descriptions parallèles (intersection de transducteurs) que séquentielles (composition de transducteurs). Outre la formalisation de descriptions à plus de deux niveaux, le second apport du langage est de proposer la notion *d'unité d'analyse* pour mettre en correspondance les différents niveaux.

La première section de cet article motive l'approche au moyen d'exemples montrant l'intérêt de représentations multiples et d'unités d'analyse diverses. La seconde section est une présentation du langage. La troisième situe ce travail par rapport à d'autres approches représentatives du domaine.

2 Intérêt des niveaux et unités multiples

2.1 Motivation de l'approche multi-niveaux

Nous allons motiver l'utilisation de descriptions linguistiques utilisant plus de deux représentations au moyen de quelques exemples.

George Anton Kiraz a proposé une description de la morphologie de la langue syriaque utilisant quatre représentations : une représentation écrite de surface et trois dimensions orthogonales représentant des unités d'analyse éventuellement discontiguës : la racine consonnantique, le schéma vocalique et un motif paradigmatique (Kiraz, 2001). Le tableau suivant offre un exemple de forme avec les quatre niveaux correspondants.

racine	k		t			b
schéma vocalique		a			e	
motif paradigmatique	C	V	C	C	V	C
forme	k	a	t	t	e	b

Ce type de description a été utilisée également pour la morphologie de l'arabe. Elle s'apparente à l'approche de la morphologie des langues sémitiques de (McCarthy, 1981). Chacun des niveaux de description est décrit au moyen d'un inventaire exhaustif. La compatibilité des différents niveaux est régie par des règles.

Prenons un autre exemple : la description des formes morphologiques utilisées dans des messages SMS. L'écriture utilisée dans ce type de messages (et d'autres formes de communication électronique) mélange plusieurs sous-systèmes, l'un d'entre eux étant l'écriture habituelle, d'autres étant phonétiques ou idéographique. Le passage d'un sous-système à un autre se fait parfois au sein d'une même forme. On ne peut pas analyser une telle forme sans recourir à deux représentations, l'une graphémique l'autre phonétique, ce qui conduit à une description à trois niveaux. Nous utilisons un quatrième niveau d'analyse qui explicite le sous-système graphique utilisé par une séquence de caractères. Voici un exemple de forme avec les niveaux correspondants. La première lettre est interprétée comme relevant du système phonétique syllabique où un caractère dénote la syllabe utilisée comme nom de la lettre, la seconde comme relevant de l'écriture habituelle et la troisième, d'une écriture habituelle dans laquelle les accents sont omis.

forme graphique	c	a	f	é
forme phonétique	k	a	f	e
translittération	k-syll		f-graph	e-graph-sa
forme de surface	k		f	e

La référence à deux logiques, l'une concernant le système graphique, l'autre le système phonétique concerne également l'écriture habituelle. Par exemple, le choix de la forme *in* ou *im* dans le préfixe négatif (im+mangeable/in+décidable) est surtout graphique, alors que le choix de la forme d'une racine verbale *peuv* ou *pouv* pour le verbe pouvoir dépend de critères phonologiques, à savoir le caractère ouvert ou fermé de la syllabe contenant la voyelle. Dans la plupart des systèmes de morphologie à deux niveaux, les deux types de phénomènes sont traités par une unique représentation qui est de nature hybride et utilise des méta-caractères pour capturer l'une ou l'autre (ou les deux) des dimensions à considérer. Si l'on pouvait décrire chaque logique séparément sans tomber dans une formalisation trop lourde, ce serait certainement préférable.

Un dernier exemple terminera cette section : celui des nombres en toutes lettres. Lauri Karttunen a publié un article sur la description des nombres en finnois (Karttunen, 2006) et il met à disposition sur la toile les fichiers contenant cette description au format des outils Xerox (xfst et lexc). Cette description relie un niveau de surface où les nombres sont écrits en toutes lettres à un niveau abstrait où une notation en chiffres arabes est concaténée à des valeurs de traits (cas, nombre, caractère ordinal ou cardinal). Cette concaténation a un caractère arbitraire puisque les deux types d'informations sont différentes. Nous préférons pour notre part une représentation à trois niveaux, avec la représentation en lettres, la représentation en chiffres et les traits. La représentation en chiffres a un intérêt propre : elle est d'usage courant. Par ailleurs, l'ordre des chiffres dans un nombre est éminemment significatif alors qu'il n'y a pas d'ordre entre les traits, au moins d'un point de vue conceptuel.

Nous sommes donc partis de la description de Lauri Karttunen pour écrire une description à trois niveaux pour les nombres en finnois et nous l'avons ensuite enrichie pour décrire également les nombres en français, avec l'ajout de nouveaux niveaux. D'une part, nous avons un niveau pour les nombres écrits en français. D'autre part, nous avons scindé les traits en trois catégories, chacune enregistrée sur un niveau : les traits communs aux deux langues (nombre et caractère ordinal/cardinal) et les traits spécifiques à une ou l'autre des langues : cas pour le finnois, genre pour le français (utilisé pour un/une, premier/première, second/seconde). Ce qui fait au total six niveaux : français, finnois, chiffres, traits communs, traits français, traits finnois. Cette description permet de traduire d'une langue à l'autre, des chiffres à une langue, de façon plus ou moins déterministe en fonction des traits renseignés. La multiplication des niveaux pourrait être un handicap si l'on approchait une taille critique pour la machine finie obtenue. Ce n'est pas le cas dans cet exemple.

2.2 Unités d'analyse

Une description morphologique nécessite de coordonner une multiplicité d'unités d'analyse. C'est déjà le cas pour les descriptions à deux niveaux habituelles. Ça l'est encore davantage avec les descriptions à multiples niveaux reliant des représentations très différentes.

La partie de la morphologie qui décrit la structure des formes fait appel à la notion de morphème (ou une autre notion d'unité dans le cas de morphologie non concaténative). Cette description ne prend pas en compte la forme de chaque morphème. Cette indifférence aux contenus des affixes est particulièrement visible dans la syntaxe du système PC-Kimmo avec la construction `ALTERNATION`, et dans une moindre mesure avec lexc de Xerox.

Lorsqu'on multiplie les représentations différentes, les correspondances entre représentations peuvent mettre en jeu plusieurs types d'unités. Par exemple, entre un nombre en lettres et un

nombre en chiffres, la correspondance n'est pas simple, comme l'illustrent les quelques cas suivants.

3	5	6	2
trois mille	cinq cent	trente	deux

1 0	6	2
mille	trente	deux

1	3	6 0
treize-cent	trente	

Toujours dans cet exemple des nombres, les structures de traits concernent un nombre dans son intégralité. On a donc deux types d'unité en jeu dans les correspondances entre représentations : la forme complète et l'unité de correspondance chiffres/lettres. De plus, dans cette dernière, on peut distinguer des unités plus petites telles que les affixes qui interviennent dans les cas de flexion. C'est plus clair dans le cas du finlandais pour lequel une déclinaison à douze cas fléchit chacun des chiffres d'un nombre. Cela existe en français surtout sur le dernier mot de la séquence, car c'est là que se fait l'accord en genre et nombre. Il y a alors 4 types d'unités, plus la notion de symbole ou caractère qui est l'unité de base dans la description. Dans cet exemple, les différentes unités sont imbriquées les unes dans les autres, ce qui autorise une représentation arborescente des analyses.

[type=card,gen=fem]			
3	4		1
trois cent	quarante	et	un e

Ce n'est pas toujours le cas : les unités d'analyse peuvent découper certaines formes de façon différentes sans qu'il n'y ait imbrication de l'une dans l'autre. Beaucoup de phénomènes de nature phonologique relèvent d'une autre organisation que la morpho-syntaxe. Quelques exemples : la réalisation du schwa et la prononciation d'une consonne finale peuvent s'expliquer avec une notion de syllabe qui peut réunir des fragments de formes fléchies différentes. A une autre échelle, au sein d'une forme, la syllabification détermine la forme de certaines racines verbales comportant des alternances et une même syllabe peut regrouper des portions d'affixes différents. Les deux notions d'affixe et de syllabe sont orthogonales, aucune n'est imbriquée dans l'autre.

La notion d'unité d'analyse que nous voyons dans ces différents exemples sert notamment à exprimer des relations de correspondances entre représentations différentes. Au sein d'une telle unité, les différentes représentations sont soit indépendantes (par exemple, `cent` est en relation avec 100, mais il n'y a pas de relation spéciale entre quelque sous-chaîne que ce soit de `cent` et une sous-chaîne de 100), soit coordonnées au moyen d'une unité plus petite enchassée.

3 Présentation du langage

3.1 Types et opérations sur les relations n-aires

Les relations rationnelles sont intrinsèquement typées par leur arité, c'est-à-dire leur nombre de composantes. Par exemple, on ne peut pas concaténer un ensemble de couples et un ensemble de triplets[1]. En pratique, un typage plus fort est intéressant pour distinguer différentes informations. Par exemple, si l'on considère une relation à trois composantes entre nombres en toutes lettres, nombre en chiffres et structures de traits, il y a trois projections différentes qui donnent une relation binaire. Si l'on s'en tient au typage minimal par arité, on peut opérer une concaténation entre une relation binaire comportant les lettres et les chiffres et une autre entre les lettres et les

[1] Sauf à réaliser une coercition de type implicite qui complète les couples pour en faire des triplets.

traits. Nous proposons une discipline différente où l'on s'interdit cela en définissant un typage plus fort.

Pour ce faire, nous allons utiliser des noms pour identifier chaque composante, comparables aux noms d'attributs des bases de données relationnelles. Un type de relation consistera en un ensemble de noms d'attributs. Ce procédé permet de ne plus utiliser l'ordre pour identifier les composantes, apportant plus de souplesse dans les notations. De plus, les noms permettent de simplifier l'expression de certaines opérations. Il n'y a pas de différences théoriques entre relations avec composantes ordonnées et relations avec composantes nommées.

Une nouvelle opération, le renommage, consiste à changer le nom de certaines composantes. Elle réalise une conversion de type explicite (cast).

Les opérations qui changent le type de leurs opérandes sont la projection et la jointure. La projection consiste à éliminer purement et simplement certaines composantes. Elle admet deux variantes syntaxiques, l'une où on spécifie ce que l'on élimine, l'autre où on spécifie ce que l'on conserve.

La jointure consiste à fusionner deux relations qui peuvent avoir des composantes en commun, mais pas nécessairement toutes (Kempe *et al.*, 2004). Les tuples du résultats proviennent de la fusion de deux tuples venant chacun d'un des opérandes et qui sont égaux sur les composantes communes. Le type du résultat est l'union des type des opérandes. Si ces deux types sont disjoints (il n'y a aucun niveau commun), la jointure est un produit cartésien. S'ils sont égaux (tous les niveaux sont communs aux deux machines), la jointure est une intersection. S'il y a une composante commune, la jointure s'apparente à une composition, mais il n'y a pas d'élimination de cette composante dans le résultat. Cette élimination systématique a pour objet de maintenir à deux le nombre de composantes du résultat, ce qui est hors de propos lorsqu'on admet des relations n-aires.

La jointure pose un problème : en général, la jointure de deux relations rationnelles n'est pas une relation rationnelle. C'est bien connu dans le cas particulier de l'intersection : l'intersection de deux relations rationnelles n'est pas toujours rationnelle. Face à cette difficulté, il convient de définir quelles jointures sont possibles à réaliser.

3.2 Expressions rationnelles étendues

Une unité est représentée par un tuple typé qui comporte un certain nombre de composantes nommées. Une composante est soit directement un des niveaux d'analyse, auquel cas elle contient une chaîne de symboles de l'alphabet, soit une composante est le résultat d'un produit cartésien imbriqué, auquel cas elle contient une chaîne de tuples ayant le même type.

Chaque expression rationnelle a un type, qui est un ensemble de niveaux. Si cet ensemble est un singleton, elle est une expression rationnelle ordinaire dénotant un langage. Si le type contient plus d'un niveau, l'expression consiste en une expression rationnelle dont les atomes sont des tuples ayant ce type. Les composantes des tuples sont des expressions rationnelles sur leurs types respectifs. Chaque tuple présent dans une description doit être une occurrence d'une unité d'analyse préalablement déclarée.

Nous allons présenter des fragments d'une grammaire décrivant la relation entre mots écrits en lettres et mots écrits en chiffres. Outre les deux niveaux des chiffres et des lettres, elle comporte un niveau comportant l'information de genre qui détermine les formes en lettres se terminant

par un/une. Il y a deux unités d'analyse : une unité de correspondance chiffre/lettre et une unité qui est le nombre. Le nombre peut comporter plusieurs unités de correspondance chiffre/lettre. Les noms des deux unités sont respectivement `c2l` et `num`. La grammaire commence avec les déclarations de l'alphabet divisé en classes de symboles, des types de structures de traits et des unités d'analyse. Dans les expressions, les symboles comportant plusieurs caractères ou des caractères de ponctuation sont écrits entourés par les signes $<$ et $>$. Par exemple, `< >` désigne un espace, `<lettre>` désigne la classe de caractère `lettre`. La chaîne vide est notée `<>`.

Le nom d'unité d'analyse est donné en début de chaque tuple. Les composantes des tuples sont nommées, ce qui permet d'accepter plusieurs syntaxes : avec les noms, l'ordre n'étant pas significatif, ou sans les noms, l'ordre étant celui de la déclaration. De plus, la déclaration comporte une valeur par défaut pour chacune des composantes, ce qui permet d'abbréger la notation.

La syntaxe exige que les noms de niveaux commencent avec `T_` (t pour *tape*, ruban). Le point d'interrogation est l'opérateur qui rend optionnelle l'expression qui précède. Les points de suspensions sont utilisés pour raccourcir l'exemple d'un point de vue typographique et ne font pas partie de la syntaxe du langage.

```
class genre is masc, fem;
class chiffre is 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;
class lettre is a, b, ..., <->, < >;
class sep is < >, <->;
unit c2l is {T_chif: <chiffre>*, T_let: <lettre>*}
unit num is {T_fs: <genre>, segs: {c2l}* }
regexp zero is {num: segs = {c2l: 0, zero}};
regexp n_1 is {c2l: 1, (< >et< >)?un(e?)};
regexp n_2_6 is {c2l: 2, <->?deux}|...|{c2l: 6, <->?six};
```

Les opérateurs binaires doivent avoir deux opérandes du même type, c'est-à-dire décrivant deux relations ayant les mêmes composantes ou niveaux. Il est ainsi possible de concaténer deux occurrences d'une unité mais impossible de concaténer par exemple un `num` et un `c2l`.

La notation par accolades dénote un produit cartésien avec ajout implicite d'accolades typées dans les composantes. Par exemple,
`{num: segs = {c2l: 0, zero}}` dénote le produit cartésien :
`(T_fs= <{num>(<masc>|<fem>)<num}>)` $\times$
`(T_chif= <{num><{c2l>0<c2l}><num}>)` $\times$
`(T_let= <{num><{c2l>zero<c2l}><num}>)`
dans lequel `<{c2l>`, `<c2l}>`, `<{num>` et `<num}>` sont quatre symboles atomiques ordinaires. Il y a donc inscription des limites d'unités dans les chaînes, ce qui a pour conséquence que l'intersection ou la jointure considèrent comme différents une même chaîne découpée de deux façons différentes.

Comme tout produit cartésien, les unités doivent agglomérer des composantes indépendantes. Cela signifie que les types des différentes composantes de l'unité doivent être disjoints. Il est par exemple interdit de définir `unit faux is {T_let: <lettre>*, suite: {num}+}`, car les deux composantes ont le niveau `T_let` en commun.

La syntaxe proposée pour les expressions rationnelles ne permet de définir que des structures arborescentes, dans lesquelles les différentes unités sont imbriquées.

3.3 Autres constructions

Nous venons de voir comment sont définies des relations rationnelles au moyen d'expressions rationnelles étendues avec des accolades. Une autre façon de définir des relations rationnelles consiste en un calcul où des opérateurs sont appliquées à des relations rationnelles préalablement définies. Ces opérateurs comportent les opérateurs utilisables dans les expressions rationnelles (concaténation, étoile, différence), plus quelques autres, notamment la projection, la jointure, le renommage, l'ajout et l'élimination d'unités. La syntaxe est la suivante.

```
let n_1_9 = union(n_1, n_2_6, n_7_9);
regexp schema is {c2l: T_let= (<letter>-<sep>)*}
                     {c2l:T_let= <>|(<sep>(<letter>*))}*;
let n_1_69 = union(n_1_9, n_10_19, n_20_69);
let seq = intersect(n_1_69,schema);
```

La jointure n'est pas toujours possible, puisqu'en toute généralité, le résultat n'est pas rationnel. Elle est possible dans les cas suivants : il y a au plus un niveau commun aux deux opérandes, ou alors il y a plusieurs niveaux communs mais tous ces niveaux sont synchronisés (découpés) de façon identique dans les deux machines. Nous n'irons pas plus loin dans l'expression de la condition exacte. On peut la trouver dans (Barthélemy, 2007). Dans tous les cas où elle est admise, le résultat de la jointure est une structure de graphe acyclique.

Cela permet de représenter certains découpages différents de certains niveaux (par exemple, syllabes et affixes), mais pas tous. Ces structures plus riches que des arborescences ne peuvent pas être décrites avec les expressions rationnelles étendues pour des raisons de lisibilité.

L'élimination d'une unité dans une relation n'est autorisée que dans le cas où cette unité ne concerne qu'un des niveaux de cette relation ou si c'est une unité englobante d'une autre unité ayant le même type. De même, l'ajout d'une unité n'est possible que si elle ne concerne qu'un niveau ou si c'est une unité englobante d'une autre unité ayant le même type. La raison de ces restrictions est discutée dans la section 3.4.

La morphologie à deux niveaux utilise des règles contextuelles pour spécifier des correspondances entre niveaux d'une relation. Ces règles sont de quatre types assez complexes (restriction de contexte, coercition de surface, etc.) pour les système à la Kimmo et des règles de réécriture pour les systèmes à la Xerox. Nous utilisons dans notre langage des règles de restriction généralisée de (Yli-Jyrä & Koskenniemi, 2004). Elles sont simples à décrire et généralisent élégamment les autres types de règles.

Un règle de Restriction Généralisée se lit comme une implication logique, si *précondition* alors *conséquence*. Les deux termes sont des expressions rationnelles étendues dans lesquelles deux occurrences du symbole spécial # délimitent le centre et les contextes gauche et droit. Prenons un exemple.

```
rule accord_cent is pattern: {num};
constraint {num: seg = {c2l}*#{c2l:< >?cents}#{c2l}*}
  =>  {num: seg = {c2l}*#{c2l: cents}#};
```

Une telle règle équivaut à une restriction de contexte. Elle peut se lire : si le mot *cents* avec un s apparaît, alors il doit être le dernier mot du cardinal. On a défini ici le centre comme étant toute l'unité `c2l`. On aurait pu dans cet exemple choisir comme centre la seule chaîne `cents`.

Une règle de restriction généralisée est un moyen de décrire une relation. C'est la sous-relation du motif déclaré par `pattern` qui respecte la contrainte décrite par la règle. Il ne s'agit pas de réécriture mais d'exprimer une contrainte pesant éventuellement sur plusieurs niveaux.

Une réécriture consiste à remplacer une chaîne à un certain niveau par une autre. On peut la réaliser au moyen des opérations suivantes : décrire une relation rationnelle entre la chaîne à réécrire et sa future valeur. Pour cela, la valeur future sera exprimée sur un niveau spécial que nous appellerons `T_fut`. Eventuellement, cette relation peut comprendre d'autres niveaux, qui ne changent pas eux-mêmes mais conditionnent la réécriture. Pour effectuer la réécriture, on réalise la jointure entre la nouvelle relation et la relation à modifier, puis par projection, on supprime le niveau à réécrire et finalement, on renomme `T_fut` pour lui donner le nom de ce niveau. La réécriture n'est pas une opération primitive à ajouter dans le langage, mais nous avons prévu une syntaxe pour l'exprimer simplement. La réécriture peut se définir aussi bien au moyen d'une expression rationnelle que d'une règle contextuelle.

3.4 Aspects techniques

Le travail présenté ici repose sur l'insertion dans les différentes chaînes de symboles spéciaux, les accolades, marquant le début et la fin des unités d'analyse. Ces symboles servent à la *synchronisation* des différents niveaux. Aucun produit cartésien n'est autorisé sans une telle synchronisation. Cela introduit une propriété, c'est que pour tout tuple d'une relation, le nombre d'occurrences d'une unité donné est le même pour tous les niveaux qu'elle concerne. Par exemple, pour toute forme analysée conformément à notre embryon d'exemple, le nombre d'unités `c2l` (resp. de symboles `<{c2l>`, de symboles `<c2l}>`) est le même dans le niveau des chiffres (`T_chif`) et dans celui des lettres (`T_let`). Cette propriété est utilisée pour garantir une clôture sous intersection (et différence ensembliste) des relations en utilisant l'algorithme de resynchronisation de (Eilenberg, 1976) sur des unités plus longues que des symboles. Les principes théoriques de cette approche sont discutés dans (Barthélemy, 2007).

La compilation de telles relations a été étudiée dans (Barthélemy, 2007). Cette compilation résulte en une machine finie qui peut être vue soit comme un automate, soit comme un transducteur. Dans la vision transducteur, les symboles ordinaires sont reconnus indépendamment alors que les accolades sont lues simultanément sur tous les niveaux qu'elles synchronisent. Les restrictions apportées sur la jointure et les suppressions/insertions d'accolades viennent de la nécessité de représenter des structures de graphe acyclique au moyen de chaînes de caractères. Cela suppose que les ordres partiels induits par les différents découpages en unités sont plongeables dans un ordre total et de plus qu'il est possible de choisir un ordre canonique préservé par les opérations rationnelles.

4 Comparaison avec d'autres langages

Comme nous venons de le voir, les relations rationnelles décrites par notre langage sont compilées en automates finis. Cela signifie que le pouvoir expressif du système est celui des automates finis. C'est une puissance moindre que celle de systèmes de règles de réécriture tels xfst, qui ont la puissance des relations rationnelles générales. Loin d'être un handicap, cette puissance moindre est la clé qui permet d'offrir un jeu d'opérations beaucoup plus riche que celles dont on dispose pour les relations rationnelles.

Outre les trois opérateurs rationnels et leurs dérivés, ces dernières offrent la composition et les projections. Toutes ces opérations sont également présentes dans notre langage, mais avec en plus la jointure naturelle, l'intersection et la différence.

Au-delà des aspects théoriques, nous pensons que le surcroît de puissance des systèmes de réécriture a peu d'impact en pratique pour les descriptions morphologiques. La plupart des descriptions peuvent probablement se traduire relativement facilement d'un langage vers l'autre. C'est ce qui s'est passé pour la description des nombres finnois traduite directement de xfst dans notre langage.

Nous pouvons détailler ce qui se produit au cours de cette traduction. D'une part, deux informations (les chiffres et les traits) concaténées sur le niveau lexical de xfst sont séparées sur deux niveaux différents de notre langage. Dans le processus de compilation de notre grammaire, les deux représentations sont également concaténées, mais pas nécessairement dans le même ordre. Par ailleurs, les unités vraiment importantes dans la descriptions sont matérialisées en xfst par des symboles ordinaires. Ainsi, la notion correspondant à l'unité `c2l` de notre exemple est notée par un # en xfst. La gestion de ce symbole (introduction, élimination) est explicite dans le code xfst. Un autre type d'unités est codé par le symbole `|`.

A un certain niveau d'analyse, les deux descriptions sont donc relativement proches. Les différences portent sur l'interface utilisateur et quelques nuances. L'outil de Xerox tend à décrire des fonctions multivaluées servant à transformer une entrée en une sortie. Notre outil a une approche plus relationnelle, où l'on met en relation de façon structurée des représentations différentes. Par ailleurs, notre cadre est plus déclaratif avec des notions plus abstraites comme la séparation de plus de niveaux ou la notion d'unité d'analyse. Nous pensons que cela contribue au confort de l'utilisation. Dans certains cas, la contrepartie peut être une moindre efficacité, le compilateur optimisant moins bien qu'un programmeur xfst averti.

Nous aurons du mal à établir une comparaison avec le seul autre système multi-niveaux que nous connaissions : wfsc de Xerox (Kempe *et al.*, 2003). Cet outil n'est pas diffusé et les publications à son sujet ne sont pas très détaillées. Il s'agit d'une boîte à outils de transducteurs multi-niveaux pondérés. Il implémente la jointure. Les niveaux ne sont pas nommés mais identifiés par leur rang. Il est plus général que notre outil car il implémente les relations rationnelles n-aires sans restrictions. En revanche, il ne propose pas de notion d'unité d'analyse. C'est un véritable outil avec un travail important sur les structures de données et les algorithmes alors que notre langage est une sur-couche externe au-dessus d'une boîte à outils d'automates finis.

5 Conclusion

Dans cet article, nous avons insisté sur le coeur de notre proposition, à savoir utiliser une notion d'unité d'analyse pour structurer des descriptions multi-niveaux. Dans sa mise en oeuvre, cette notion ressemble à celle d'enregistrement dans les langages de programmation.

Nous n'avons pas eu la place de présenter d'autres aspects intéressants de notre langage. La notion de sous-unité permet de simplifier l'écriture d'expressions rationnelles au moyen de déclarations donnant des valeurs par défaut à certains membres d'une unité. Par ailleurs, les structures de traits typées non récursives et à domaines de valeurs finis sont proposées dans la syntaxe et compilées en automates finis.

Notre modèle offre quelques sources de complexité qui risquent d'aller contre son objectif d'er-

gonomie et de déclarativité. Par exemple, les conditions d'emploi de la jointure ne sont pas simples. Nous espérons que beaucoup d'applications ne sont pas sujettes aux subtilités de cette question. Remarquons au passage que les règles contextuelles à la Kimmo et à la xfst ne sont pas exemptes de complexité non plus.

L'utilisation de structures arborescente a été testée au moyen de multiple exemples de taille significative. Ce n'est pas encore le cas des structures de graphes acyclique plus complexes. Leur intérêt pratique reste à évaluer.

Notre modèle est dans la lignée de la morphologie à deux niveaux en privilégiant l'aspect relationnel et l'opération d'intersection. Il s'affranchit des règles complexes et générant des conflit de cette approche. En revanche, il intègre des règles de réécriture légères venant du modèle séquentiel. Il s'inscrit dans la continuité de ses devanciers, les différences étant dans le processus de compilation pour atteindre un résultat voisin. Il apporte un plus grand confort dans le cas où les représentations sont multiples et très différentes, les autres modèles étant des solutions plus légères dans le cas où il s'agit de relier deux représentations voisines, comme une représentation canonique des affixes et une réalisation qui en est une simple alternance.

Références

ANTWORTH E. L. (1995). *User's Guide to PC-Kimmo Version 2*. Dallas, Texas : Summer Institute of Linguistics.

BARTHÉLEMY F. (2007). Multi-grain relations. In *Implementation and Application of Automata, 12th International Conference (CIAA)*, p. 243–252, Praque, Czech Republic.

BARTHÉLEMY F. (2007). Using mazurkiewicz trace languages for partition-based morphology. In *ACL*, Prague (Czech Republic).

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

EILENBERG S. (1976). *Automata, Languages, and Machines*. Orlando, FL, USA : Academic Press, Inc.

KARTTUNEN L. (2006). Numbers and finnish numerals. *SKY Journal of Linguistics*, **19**, 407–421. Festschrift in Honour of Fred Karlsson on his 60th Birthday.

KEMPE A., BAEIJS C., GAÁL T., GUINGNE F. & NICART F. (2003). WFSC – A new weighted finite state compiler. In *Proc. 8th Int. Conf. on Implementation and Application of Automata (CIAA'03)*, LNCS 2759, p. 108–119, Santa Barbara, CA, USA : Springer Verlag.

KEMPE A., CHAMPARNAUD J.-M. & EISNER J. (2004). A note on join and auto-intersection of *n*-ary rational relations. In B. WATSON & L. CLEOPHAS, Eds., *Proc. Eindhoven FASTAR Days*, Eindhoven, Netherlands.

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

MCCARTHY J. J. (1981). A prosodic theory of nonconcatenative morphology. *Linguistic Inquiry*, **12**, 373–418.

YLI-JYRÄ A. M. & KOSKENNIEMI K. (2004). Compiling contextual restrictions on strings into finite-state automata. In *Proceedings of the Eindhoven FASTAR Days 2004 (September 3–4)*, Eindhoven, The Netherlands.