
Spécifications du format des règles pour la construction d'automates de reconnaissance d'expressions

Romarc Besancon <romarc.besancon@cea.fr>

Copyright © 2005 Romarc Besançon - CEA-LIST

Revision History

Revision 0.1

7 fév 2005

RB

reprise du document de spécification préliminaire (en LaTeX)

Table of Contents

Objectif	1
Spécifications des règles	1
Besoin d'expressivité des règles	1
Format des Règles	3
formalisme des automates décrivant le contexte	3
Spécification formelle des règles	8
Limites	9
Exemples de règles	9

Objectif

L'objet de ce document est la définition du format déclaratif de règles permettant de reconnaître des expressions dans un texte analysé (en cours d'analyse). Cette reconnaissance s'appuie sur un format d'expressions régulières qui permet la construction d'automates pour la reconnaissance d'expressions particulière. Ces automates sont utilisées lors du traitement linguistique pour

- la reconnaissance des expressions idiomatiques
- la reconnaissance d'entités spécifiques comme les nombres, les dates et les entités nommées
- l'extraction de relations de dépendance pour l'analyse syntaxique

Spécifications des règles

Besoin d'expressivité des règles

Les éléments nécessaires à la définition des règles pour la reconnaissance d'expressions particulières spécifiques sont :

- Un *élément déclencheur* : c'est l'élément qui lancera le processus de reconnaissance lorsqu'il apparaît dans un texte ; il sera choisi comme l'élément le moins fréquent ou le plus caractéristique de l'expression considérée, pour éviter de trop nombreux déclenchements qui n'aboutissent pas. Le déclencheur ne doit pas forcément être au début de l'expression.
- Les *contextes gauche et droit* qui définissent l'expression autour du déclencheur (on utilise les termes "gauche" et "droit" pour désigner les contextes "précédent" et "suivant" : c'est aussi comme cela que ces termes doivent être compris dans une langue comme l'arabe) : ces contextes seront

Spécifications du format des règles
pour la construction d'automates
de reconnaissance d'expressions

définis en utilisant un formalisme proche des expressions régulières. Les éléments dont on a besoin pour définir ces contextes sont les suivants :

- Des unités de reconnaissance : elles peuvent être virtuellement n'importe quelle propriété (ou combinaison de propriétés) associée à un élément unitaire de la phrase en sortie de l'analyse morphologique ou après désambiguïsation (un *FullToken* ou un *DicoWord*). En pratique, les propriétés qui nous semblent a priori intéressantes pour la définition des règles sont les suivantes :
 - Des mots simples : la reconnaissance se fait alors sur la simple forme de surface (la forme directe du mot dans le texte ou l'une de ses variantes orthographiques obtenue lors de l'analyse morphologique) ;
 - Des catégories grammaticales seules : par exemple pour prendre en compte l'insertion d'un ou plusieurs adjectifs ;
 - Des formes normalisées de mots : la correspondance se fera alors avec toutes les formes fléchies du mot (la catégorie grammaticale du mot doit alors être obligatoirement spécifiée : par exemple "porte" n'acceptera pas les mêmes flexions s'il est nom ou verbe) ;
 - Des classes, qui regroupent un certain nombre de mots ou d'éléments particuliers (définis dans une liste) qui partagent une propriété qui n'est pas identifiée dans le dictionnaire (ce peut être le cas pour des annonceurs de noms propres, par exemple) ;
 - d'autres propriétés peuvent également être envisagées, comme des traits sémantiques ou morphologiques (initiale en majuscule, par exemple), ou des contraintes sur des valeurs numériques ;
- Des opérations sur ces unités (opérations classiques des expressions régulières) :
 - Des groupements de mots : par exemple "de la" peut être considérée comme un groupe à manipuler comme une unité ;
 - Des alternatives d'unités ou de groupes : "M." ou "Monsieur" ;
 - Des cardinalités sur l'occurrence d'une unité ou d'un groupe : on peut introduire un ou deux adverbes dans une forme verbale composée, mais pas plus de trois. On peut également avoir besoin de spécifier une cardinalité non limitée (la limite pourra alors être la limite de la phrase ou du texte, selon l'usage).
 - Des unités ou groupes optionnels : cette propriété est un cas particulier de cardinalité (au moins 0, au plus 1 occurrence), mais on peut la garder pour faciliter l'écriture ;
 - la négation d'une unité ou d'un groupe : on peut vouloir exprimer qu'une unité est reconnue si elle n'a pas une certaine propriété (par exemple, n'importe quel mot sauf un point).
- Le *type* de l'expression reconnue ;
- La *forme normalisée* de l'expression (pour certaines entités dont la forme normalisée doit être calculée, comme les nombres ou les dates, un code permettant d'indiquer le type de normalisation peut également être envisagé) ;
- Les *bornes de l'expression* reconnue ou l'indication de quels mots parmi ceux reconnus ne font pas partie de l'expression reconnue : les déclencheurs et le contexte peuvent en effet permettre de reconnaître ou de caractériser une entité, sans faire partie de cette entité ;
- Des *contraintes* supplémentaires sur certains éléments reconnus ou entre certains éléments de la règle doivent également pouvoir être spécifiés (par exemple des contraintes d'accord en genre, en nombre, en personne etc.)
- l'indication possible du *mot principal* de l'entité reconnue (tête de l'expression). Cette indication est particulièrement utile pour les expressions idiomatiques. Une expression idiomatique peut en

effet être fléchi, et les propriétés linguistiques associées à l'expression dans son ensemble seront alors celles de la tête de l'expression (par exemple, la forme pronominale "se trompait" devra être reconnue comme une forme verbale à l'imparfait de l'indicatif, propriétés linguistiques du mot "trompait", tête de l'expression).

- l'indication de la *relativité d'application* de la règle : également utile pour les expressions idiomatiques, cette indication doit permettre d'indiquer si la reconnaissance de la règle est absolue (par exemple "au fur et à mesure" est *toujours* une expression idiomatique) ou est une possibilité à désambiguïser dans une phase ultérieure du traitement ("rendez-vous" peut être un mot composé ou un verbe suivi d'un pronom).
- la possibilité d'indiquer la négation d'un type: si la règle s'applique, alors aucune autre règle du type indiqué ne devra être appliquée (permet de mieux structurer les règles et de contourner ou rattraper facilement certains problèmes).

Format des Règles

Pour répondre aux besoins d'expressivité des règles spécifiés dans la section précédente, le formalisme de règles utilisé actuellement est le suivant :

Le séparateur principal de la règle est le signe ":" Une règle est définie par une expression du type :

`<déclencheur>:<contexte gauche>:<contexte droit>:<type d'expression>:<forme normalisée>`

Les `<contexte gauche>` et `<contexte droit>` utilisent le formalisme définis dans la section suivante.

Les contraintes sont ajoutées à la suite de ces règles.

formalisme des automates décrivant le contexte

Unités simples

Unités	azer
Mots simples	mot
Catégories grammaticales seules	\$CG
Formes normalisées de mots	lemme\$CG
Propriétés morphologiques	t_xx ou T_xx
Classes	@classe
N'importe quel mot	*

Catégories grammaticales

les catégories grammaticales sont définies en utilisant les codes symboliques internes au système (type L_NOM_COMMUN). La catégorie peut spécifier seulement une macro-catégorie ou une paire macro-catégorie/micro-catégorie séparées par un tiret. exemples: \$L_NC, \$L_NC-L_NC-GEN sont des spécifications correctes de catégories.

Classes

Les classes sont définies explicitement dans le fichier des règles ou un fichier externe, par une liste des éléments de la classe. La syntaxe de la définition d'une classe de mots est simplement @classe=(unit1,unit2,unit3,...) Les éléments peuvent être n'importe quelle unité définie dans la section précédente. La virgule est le séparateur entre les éléments, il n'y a pas d'espace après la virgule, par contre, des retours à la ligne sont autorisés. Pour l'analyse syntaxique, la définition de classes de catégories grammaticale est possible.

Propriétés morphologiques

Les propriétés morphologiques sont identifiées par les types donnés aux unités linguistiques par le tokeniseur (si les types donnés en sortie du tokeniseur changent, ces formalismes des propriétés morphologiques devront être modifiés en conséquence).

Les noms des types sont préfixés (par convention, dans le tokeniseur) par "t_" : ce préfixe est utilisé comme pour reconnaître directement les noms des types dans les règles. Ces types correspondent aux entrées "<default>" possibles dans l'automate du tokeniseur.

A titre d'exemple, les types définis par le tokeniseur du français sont les suivants :

t_acronym	t_alphanumeric	t_capital	t_capital_1st
t_capital_small	t_cardinal_roman	t_comma_number	t_dot_number
t_fraction	t_integer	t_ordinal_integer	t_ordinal_roman
t_pattern	t_sentence_brk	t_small	t_word_brk

Les types définis par les tokeniseur des autres langues sont également utilisables de manière transparente avec cette notation.

Une notation structurée des types du tokeniseur est également possible (cette notation est considérée obsolète parce qu'elle ne permet pas de prendre en compte certains types introduits par les nouvelles langues comme l'arabe et le chinois, mais elle est encore fonctionnelle et est encore utilisée dans les fichiers de règles, c'est pourquoi sa spécification est conservée ici).

T_A-	alphabétique	T_N-	numérique
T_Ac-	alphabétique majuscules	T_Ni	numérique entier
T_As-	alphabétique minuscules	T_Nc	numérique avec virgule
T_A1-	alphabétique première lettre majuscule	T_Nd	numérique avec point
T_Aa-	alphabétique acronyme	T_Nf	numérique fraction
>T_Am-	alphabétique majuscules et minuscules	T_No	numérique ordinal
T_A-c	alphabétique chiffres romains cardinal	T_U	alphanumérique
T_A-o	alphabétique chiffres romains ordinal	T_P	pattern
T_A-n	alphabétique non chiffres romains	T_W	word break
		T_S	sentence break

Indication de la tête d'une expression

La tête d'une expression sera indiquée par le caractère >& précédant l'unité identifiée comme la tête.

Exemple:

abondance:&corne\$L_NC d':::IDIOM\$Ncfs:corne d'abondance

Contraintes sur les valeurs numériques

de telles contraintes sont en particulier utiles pour la reconnaissance des dates. Elles portent sur les formes numériques des nombres et permettent d'indiquer la valeur précise souhaitée ou l'intervalle de valeurs souhaité (entre m et n), avec la notation suivante : $T_Ni=n$ ou $T_Ni>m<n$.

Note: Ces contraintes devraient être traitées comme des contraintes supplémentaires sur une unité (elles permettraient de porter sur des unités d'un type autre que numérique).

Opérations sur les unités simples

Dans le tableau suivant donnant les opérations possibles, les éléments indiqués "elt" sont soit des mots, soit des groupes (séquences ou alternatives).

Opérations	
séquence (l'un après l'autre)	(elt1 elt2 ...)
alternative (l'un ou l'autre)	(elt1 elt2 ...)
élément optionnel	elt?
cardinalités sur l'occurrence d'un élément : entre i et j fois	elt{i-j}
cardinalités sur l'occurrence d'un élément : entre i et un nombre infini de fois	unité{i-n} ou (...){i-n}
négation d'une unité (la négation d'un groupe n'étant pas traitée, elle est à éviter)	^unité

Bornes de l'expression

L'indication de quelles portions d'expression ne font pas partie de l'expression reconnue est donnée en encadrant ces portions d'expressions par des crochets [...]. Ces crochets doivent être placés autour des unités ou groupes complets (en particulier, ils englobent aussi les modificateurs de groupes indiquant l'optionnalité d'un groupe).

Le déclencheur peut ne pas faire partie de l'expression reconnue (on placera de la même façon des crochets autour du déclencheur).

Type de l'expression

La liste des types possibles sont définis dans un fichier externe. Actuellement, les types d'expressions reconnues sont tous mis dans le même fichier, qu'elles soient entités nommées, expressions idiomatiques, relations de dépendance...

Le champ du type de l'expression peut également contenir des informations supplémentaires:

- des propriétés linguistiques associées à l'expression reconnue: ces propriétés sont utiles lorsque la reconnaissance de l'expression doit permettre la création d'un nouveau token (c'est le cas pour les expressions idiomatiques). L'association de propriétés linguistiques se fait en ajoutant le symbole "\$" après le type de l'expression, suivi du code des propriétés linguistiques (ce code doit être le code numérique, mais des scripts de compilation permettent de prendre en compte le code symbolique à la Grace, comme IDIOM\$Ncms).
- la relativité d'application de la règle (également utile pour les expressions idiomatiques) : l'ajout de *ABS_* devant le type de l'expression indique que cette expression est absolue (elle est toujours vraie quelle que soit le contexte).
- la négation du type: l'ajout de *NOT_* devant le type de l'expression permet d'indiquer que si la règle s'applique, aucune autre règle du type indiqué de devra être appliquée avec ce déclencheur.

Spécifications du format des règles
pour la construction d'automates
de reconnaissance d'expressions

Les types d'expressions définis pour les entités nommées sont les suivants :

NUMEX	pour les nombres et les mesures
TIMEX	pour les dates
PERSON	pour les noms de personnes
LOCATION	pour les noms de lieux
ORGANIZATION	pour les noms d'organisations
PRODUCT	pour les noms de produits
EVENT	pour les événements

Pour les expressions idiomatiques, un seul type est défini: le type IDIOM.

Pour l'analyse syntaxiques, les types définis correspondent aux différents types de relations de dépendance considérés. Voici une liste non exhaustive de ces relations :

DETSUB	Relation entre déterminant et substantif (le -> chat)
ADJPRESUB	Relation entre adjectif prénominal et substantif (beau->chat)
COMPADJ	Complément d'adjectif
COMPADV	Complément d'adverbe
ADVADJ	Relation entre adverbe et adjectif
ADVADV	Relation entre deux adverbes
SUBADJPOST	Relation entre un substantif et un adjectif postnominal (chat <- noir)
COMPDUNOM	Relation de complément du nom (chat <- Pierre, dans "chat de Pierre")
SUBSUBJUX	Deux substantifs juxtaposés en français
TEMPCOMP	Temps composé

Forme normalisée de l'entité

La forme normalisée de l'expression est simplement donnée comme une chaîne de caractères. Pour les types dont la forme normalisée doit être calculée, un code peut être donné, indiquant la façon de dont cette normalisation doit être faite. Les codes actuellement définis sont les suivants :

N_DATE	normalisation des dates en jour, mois, année
N_NUMBER	normalisation des nombres (calcul de la valeur de nombres en toutes lettres)
N_PERSON	normalisation des noms de personnes en nom, prénom, titre

Dans une version ultérieure des règles, ces normalisations devraient utiliser les actions.

Contraintes et actions

Des contraintes ou des actions peuvent être attachées aux règles de reconnaissance d'expressions. Les contraintes peuvent porter sur un ou deux éléments de la règle: elles correspondent à une fonction qui sera appelée avec en argument le ou les deux noeuds des éléments correspondant et qui renvoie un

booléen. Les actions sont des fonctions appelées à la fin de l'application de la règle, en fonction du succès de la règle (est-ce que l'expression a été reconnue ou non). Elles n'utilisent pas les éléments de la règle. Elles peuvent éventuellement utiliser le résultat produit par la règle (ceci est décidé à l'écriture de la fonction, et n'est pas indiqué directement dans la règle).

Par souci de lisibilité des règles, les contraintes entre des éléments d'une règle et les actions sont exprimées à l'extérieur de la définition des règles.

A la suite des règles (ou dans les lignes suivantes), les contraintes commencent par un +, et s'écrivent :

```
+constraintName(elt1,elt2,"complement")
```

ou

```
+constraintName(elt,"complement")
```

Dans le premier cas, les *elt1* et *elt2* sont les éléments sur lesquels portent la contrainte, le complément (entre guillemets), est optionnel et peut être utilisé pour passer une information supplémentaire à la fonction.

Dans le second cas, *elt* est le seul élément sur lequel porte la contrainte.

Les éléments sont repérés par leur position dans la règle, en deux temps : d'abord, le contexte, qui peut être *right* (contexte droit), *left* (contexte gauche) ou *trigger* (déclencheur), puis la position du mot dans le contexte (l'indication des positions des mots est pour le moment très limitée: on peut seulement accéder à des éléments simples dans le contexte: les groupes de mots sont comptés comme un élément)

Exemple: pour le cas des verbes pronominaux, si l'on veut faire la distinction entre "je m'arrête", "je t'arrête", "tu m'arrêtes", "tu t'arrêtes") : on peut alors utiliser une règle intégrant des contraintes d'accord :

```
arrêter$L_V:$L_PRON-L_PRON_REFLEXIF @PronPrev[$?]::IDIOM$V:s'arrêter  
+AgreementConstraint(trigger.1,left.1,"PERSON")  
+AgreementConstraint(trigger.1,left.1,"NUMBER")
```

où *@PronPrev* est la classe des catégories des pronoms personnels préverbaux.

Les actions sont définies à la suite des règles (ou dans les lignes suivantes), par un =, suivi d'un signe > ou < et du nom de la fonction.

- =>faitQuelqueChose() indique que l'action *faitQuelqueChose* sera effectuée en cas de succès d'application de la règle;
- =<faitAutreChose() indique que l'action *faitAutreChose* sera effectuée en cas d'échec d'application de la règle;

Comme pour les contraintes, des compléments peuvent être passés à la fonction.

Compléments de syntaxe des fichiers de règles

L'utilisation d'un caractère d'échappement (\) permet d'introduire dans la définition des unités les caractères de la syntaxe " :()[]{}^|@\$&", sans qu'ils soient interprétés.

D'autre part, pour permettre de rendre les fichiers de règles plus structurés et plus lisibles, les éléments de syntaxe suivants ont également été définis :

- les lignes commençant par " #" sont des commentaires (un " #" qui n'est pas en début de ligne n'est pas interprété comme un commentaire) ;

- la primitive *"include"* suivie d'un nom de fichier (ou de plusieurs noms de fichiers séparés par des virgules) permet d'inclure des fichiers de règles externe (ces fichiers sont interprétés de façon complètement indépendante: par exemple, les classes définies dans les fichiers inclus ne sont pas accessibles dans le fichier incluant, ni le contraire);
- la primitive *"use"* suivie d'un nom de fichier (ou de plusieurs noms de fichiers séparés par des virgules) permet d'inclure des définitions de classes de mots externes: les classes
- une indication de l'encodage du fichier peut être faite au début du fichier, par la primitive *"define encoding="*. Les seuls codages possibles sont pour l'instant "latin1" et "utf8". Le codage par défaut est "latin1" (l'indication doit être placée avant la première ligne qui pourrait être mal codée).
- dans le but de rendre plus générique l'application des règles, il est possible d'associer une action par défaut à toutes les règles d'un fichier. Cela se fait avec la primitive *"set defaultAction="* suivie d'un nom d'action.

Spécification formelle des règles

La grammaire EBNF décrivant la définition des règles est présentée dans cette section.

```

<définition>      ::= { <règle> }
<règle>           ::= <déclencheur> ":" <contexteGauche> ":"
                    <contexteDroit> ":" <type> ":" <formeNormalisée>
                    [<contrainte>]* [<action>]*
<déclencheur>     ::= ["[" <unitéSimple> "]" ]
<contexteGauche>  ::= { <élément> }
<contexteDroit>   ::= { <élément> }
<type>            ::= <Chaîne> [<catégorie>]
<formeNormalisée> ::= <Chaîne>
<contrainte>      ::= "+" <nomFonction> "(" <eltIndex> [, <eltIndex>]
                    [, "\" <Chaîne> "\" ] ")"
<action>          ::= "=" <actionAppl> <nomFonction>
                    "(" ["\" <Chaîne> "\" ] ")"
<nomFonction>     ::= <Chaîne>
<actionAppl>      ::= ">" | "<"
<eltIndex>        ::= <part> "." <index>
<part>            ::= "trigger" | "left" | "right"
<index>           ::= <Entier>
<élément>         ::= ["[" [<modifieurPre>] <unitéComplexe>
                    [<modifieurPost>] "]" ]
<unitéComplexe>   ::= <unitéSimple> | <groupe> | <alternative>
<groupe>          ::= "(" <unitéComplexe>* ")"
<alternative>     ::= "(" <unitéComplexe> ("|" <unitéComplexe>)+ ")"
<unitéSimple>     ::= ["&"] <motGénéralisé>
<motGénéralisé>   ::= <motSimple> [ <catégorie> ] | <catégorie> |
                    <classe> | <Tstatus>
<modifieurPre>    ::= "^"
<modifieurPost>   ::= "?" | "{" <cardinalité> "-" <cardinalité> "}"
<cardinalité>     ::= <Entier> | 'n' | 'N'
<motSimple>       ::= <ChaîneSansBlanc> | "*"
<catégorie>       ::= "$" <Chaîne>
<classe>          ::= "@" <Chaîne>
<Tstatus>         ::= "t_" <Chaîne> | T_ <propriétéMorphologique>
<propriétéMorphologique> ::= 'A' <MorphoAlphaCap> <MorphoAlphaRoman> |
                             'N' <MorphoNumeric> | 'U' | 'P' | 'W' | 'S'
<MorphoAlphaCap>  ::= 'c' | 's' | 'l' | 'a' | 'm'

```

<MorphoAlphaRoman>	::= 'c' 'o' 'n'
<MorphoNumeric>	::= 'i' 'c' 'd' 'f' 'o'

Limites

Limites d'expressivité des règles

Le formalisme défini ici ne permet pas d'exprimer :

- le groupement de plusieurs propriétés pour une seule unité : le formalisme actuel ne permet pas de spécifier qu'une unité doit être de plusieurs types à la fois (on ne peut pas exprimer par exemple qu'une unité doit être un mot commençant par une majuscule et un nom propre). On peut seulement ajouter ça dans des contraintes extérieures.
- la négation d'un groupe de mots : la négation d'un groupe complexe de mots (succession ou alternative) introduit des problèmes supplémentaires car elle ne se traduit pas simplement par un distribution de la propriété de négation sur chacun des éléments (ce que fait l'implémentation actuelle) :
 - pour les alternatives : l'expression *not(a/b)* est interprétée comme *(not(a)/not(b))*, ce qui est faux. L'implémentation de cette fonction demande donc également la conjonction de propriétés au niveau d'une unité ;
 - pour les groupes : l'expression *not(a b)* est interprétée comme *(not(a) not(b))* alors qu'elle devrait être interprétée comme *(not(a b)/a not(b))*.
- La définition de sous-automates serait un outil utile pour le développement des règles, mais n'est pas implémentée.

Exemples de règles

Règles de reconnaissance des entités nommées

Voici un exemple de quelques règles simples pour la reconnaissance des noms de journaux français :

```
Libération:::ORGANIZATION:
Monde:Le:Diplomatique:ORGANIZATION:
Monde:Le:de l'Education:ORGANIZATION:
Monde:Le:::ORGANIZATION:
Courrier::International:ORGANIZATION:
Canard::Enchaîné:ORGANIZATION:
```

Un autre exemple de règles, plus complexes, pour la reconnaissance des noms de personnes :

```
@Firstname:[ (@Title|@FunctionTitle)? ]: ((de|da|le)? T_A1){1-2}: PERSON:N_PERSON
T_A1:[ (@Title|@FunctionTitle) ]: T_A1{0-2}: PERSON:N_PERSON
T_A1:(T_A1|T_Amh){0-2}:, @FunctionTitle: PERSON:N_PERSON
```

La première de ces règles se déclenche sur un prénom (la liste des prénoms est explicitement définie dans le fichier des règles), le contexte gauche contient, de façon optionnel, un titre (M., Mme, Dr, ...) ou une fonction (président, député,...), qui n'est pas gardé dans la règle finale ; le contexte doit est

composé d'un ou deux mots commençant par une majuscule, éventuellement précédé de " de ", " da " ou " le ".

La seconde reconnaît les noms de personnes introduits par des titres ou des noms de fonctions, mais sans indication de prénom.

La troisième reconnaît des noms de personnes dont la fonction est mise en apposition postérieure (par exemple, " Sanjiv Sidhu, président d' i2 Technologies ").

Voici un autre exemple de règles, pour la reconnaissance de dates, déclenchées sur les noms de jours ou de mois :

```
# lundi 22 mai 1968
$L_NC-L_NC_JOUR::T_Ni>1<31 $L_NC-L_NC_MOIS (T_Ni>1000<3000|T_Ni>1<99)?::TIMEX:N_
# 22 mai 1968
# 18 juin 40
# 31 octobre prochain
$L_NC-L_NC_MOIS:(T_Ni>1<31)?:(prochain|dernier|suivant|(T_Ni>1000<3000|T_Ni>1<9
```

règles de reconnaissance des expressions idiomatiques

Voici des exemples de règles pour la reconnaissance des expressions idiomatiques du français :

```
&arrêter$L_V:$L_PRON-L_PRON_REFLEXIF @PronPrev[$?]::IDIOM$V:s'arrêter
+AgreementConstraint(trigger.1,left.1,"PERSON")
+AgreementConstraint(trigger.1,left.1,"NUMBER")
fur:au:et à mesure (de|d'):IDIOM$Sg:au fur et à mesure de
fer:&rideau de::IDIOM$Nc:
```

où \$Sg indique une préposition générale.