SYGMART [1] : manuel de référence
Version 5.4

J. Chauché

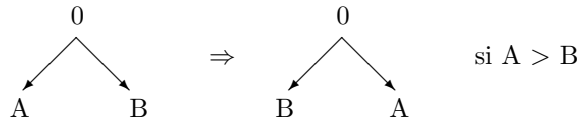
Janvier 2008

[1]Copyright: **SyText**

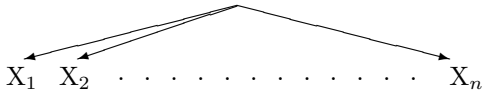
Table des matières

INTRODUCTION

Un traitement algorithmique suppose la définition d'objets et la manipulation de ces objets. Tout traitement se réduit à un traitement de chaîne à chaîne: une chaîne de caractères définit l'entrée du traitement, et, la chaine déduite, le résultat de ce traitement. Entre ces deux chaines le traitement s'effectue sur des objets intermediaires suivant des processus définis préalablement. Deux approches sont possibles pour ce traitement intermediaire. La première met l'accent sur la définition procédurale du traitement. Cette approche dynamique qui transparait à travers la plupart des langages de programmation ne permet pas une approche globale d'un univers donné puisqu'elle appréhende cet univers que par la possibilité du parcours ou de la construction d'un élément. Toute la réflexion est alors centrée dans ce type d'approche sur ces questions essentielles de parcours et de construction. La deuxième approche possible met au contraire l'accent sur l'objet manipulé. Cet objet n'a pas à être construit puisqu'il est toujours présent et seul sa modification entre dans le processus. De même son parcours n'est qu'exceptionnel puisqu'un objet est apprehendé globalement. Un premier exemple de cette différence d'approche peut être donné par l'étude classique des tris. Une suite d'éléments doit être ordonnée et diverses méthodes bien connues peuvent être utilisées pour réaliser cette opération. Dans la méthode dite 'Bubble sort' les éléments sont placés dans un tableau qui est parcouru jusqu'à ce qu'il soit ordonné. Au cours de ce parcours une comparaison et éventuellement un réarangement est éffectué entre deux éléments voisins. Deux boucles imbriquées de parcours sont donc nécessaires pour réaliser cet algorithme. Si l'on effectue le même programme dans le système TELESI une seule règle est nécessaire pour réaliser cette opération:



La structure d'entrée est constituée de la liste des éléments:



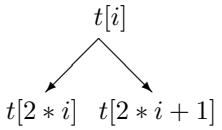
La règle ainsi définie sera appliquée de façon itérative et définira le tri suivant cette méthode. L'écriture complète est la suivante:

&GRAM: BUBBLE(I). /∗ application itérative de la grammaire ∗/

ECH: 0(A,B) // F(A) > F(B) ⇒ 0(B,A).

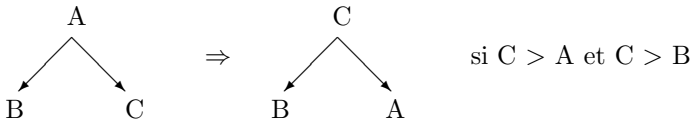
Le processus s'arrêtera lorsque la règle ne sera plus applicable et donc lorsque la suite sera triée.

La méthode du 'Heap sort' est plus élaborée et peut constituer un deuxième exemple. Cette méthode repose sur une procédure récursive de remise en tas. Un tableau est considéré comme une arborescence binaire et est en tas lorsque cette arborescence possède la propriété d'avoir toutes ces sous arborescences en tas; c'est à dire que la racine de chaque sous arborescence est supérieure à ces deux descendants:



$t[i] > t[2 * i]$
et
$t[i] > t[2 * i + 1]$

Cette procédure de mise en tas utilise un cheminement dans le tableau. La programmation de cet algorithme simule ce cheminement, elle comporte trois règles TELESI correspondant aux trois cas possibles. Une de ces règles est définie de la façon suivante:



L'écriture complète de cette procédure est la suivante:

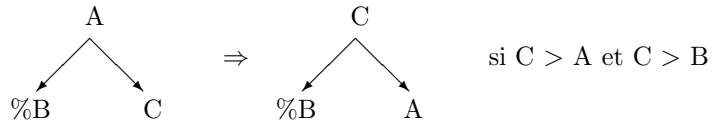
&GRAM: MTAS(U). /∗ application unique (1 seule fois) ∗/

/∗ cas d'une feuille unique ∗/
RU: 0(∗,1,∗) // F(1) > F(0) ⇒ A(B) /A:1; B:0.

/∗ descente à droite ∗/
RDAD(MTAS;B): 0(1,2) // (F(1) < F(2)) & (F(0) < F(2)) ⇒
A(1,B(∗2∗)) / A:2; B:0.

/∗ descente à gauche ∗/
RDAG(MTAS;B): 0(1,2) // (F(1) > F(2)) & (F(0) < F(1)) ⇒
A(B(∗1∗),2) / A:1; B:0.

La modification successive de l'objet manipulé (ici l'arborescence binaire) correspond au cheminement précédent défini par la procédure classique. Il est possible de définir le même cheminement en une seule règle en utilisant la notion de point facultatif:



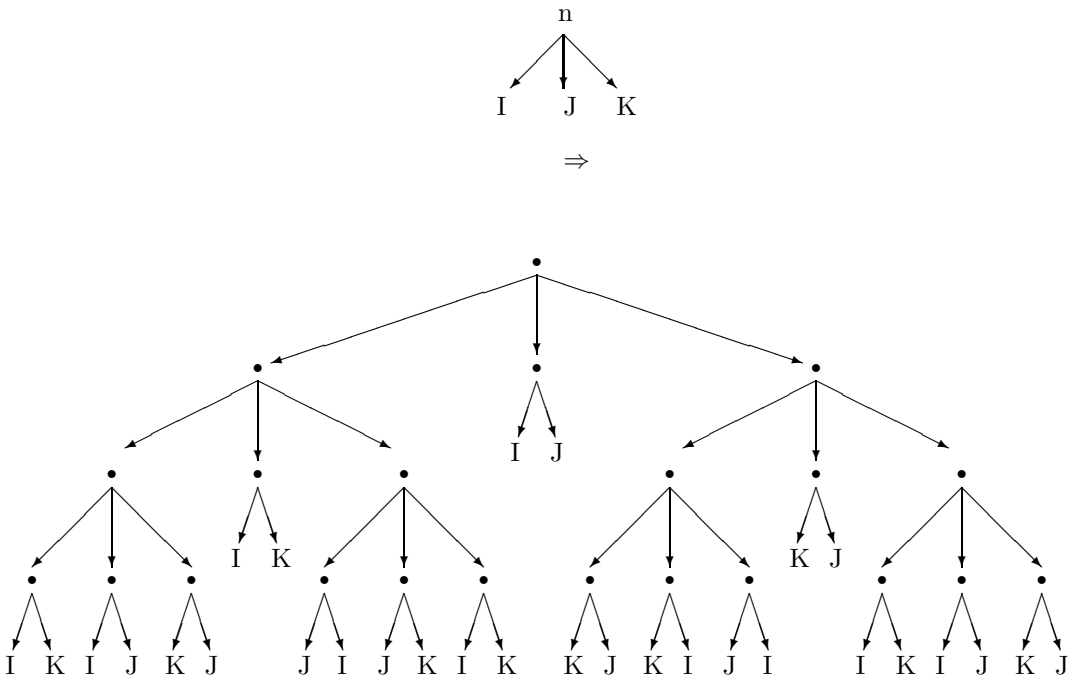
le "%" signifie que la présence du point est facultative; les points B et C sont non ordonnés.

L'écriture de cette procédure devient alors:

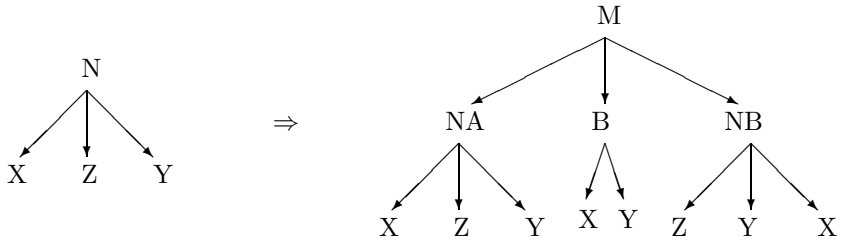
&GRAM:MTAS(U).

RTAS(MTAS;B): ∗(0(1-%2)) // (F(0) < F(1) & (F(1) > F(2) ⇒
A(∗0∗,B(∗1∗),%2) / A:1; B:0.

Un troisième exemple peut être défini par la procédure non moins classique des tours de Hanoi. Dans cette procédure récursive les différents appels définissent les mouvements à effectuer. Dans le cas de la programmation TELESI il est nécessaire de définir la forme du résultat. Supposons que nous prenions comme objets à atteindre l'arborescence constituée des différents mouvements à effectuer. La forme initiale étant constituée simplement d'une arborescence ayant dans sa racine le nombre de disques à transférer et dans chacune de ces trois feuilles le nom des tours. La transformation devient alors la suivante:



Cette transformation nécessite deux règles, une générique et une terminale. La règle générique est la suivante:



la règle finale:

L'écriture complète de la grammaire est alors la suivante:

&GRAM: HANOI(U).

RGEN(HANOI;NA/HANOI;NB): N(X,Y,Z) / N: VL > 1 ⇒
M(NA(X,Z,Y),C(X,Y),NB(Z,Y,X)) /
NA: (VL = VL(N)-1); NB: (VL = VL(N)-1).

RFIN: N(X,Y,Z) / N: VL = 1 ⇒ M(X,Y).

Le traitement automatique des langues naturelles peut être abordé suivant ces deux aspects: aspect procédural et aspect transformationnel. Une présentation de ce traitement permet donc de situer le système SYGMART par rapport aux diverses approches possibles. Bien sûr l'application de ce système

ne se limite pas à cette application mais elle constitue à l'heure actuelle une de celle qui demande le plus la maitrise d'objets complexes et variés. Le traitement automatique des langues naturelles sur ordinateur nécessite un ensemble de manipulations non numériques. Ces manipulations ont pour but essentiel la reconnaissance de relations entre les diverses parties d'un texte. Diverses théories linguistiques établissent des relations soit de façon constructive soit de façon interprétative. Certaines permettent la définition d'un modèle logique associé, et en consequence, la construction d'un modèle de traitement. Cette dernière approche a l'inconvénient de définir un outil algorithmique trop dépendant de la théorie linguistique particulière. A l'opposé, certaines théories sont trop vagues dans leurs définitions et utilisent un modèle logique dans le but de préciser les différentes notions introduites et leurs rapports. Cette dernière approche ne peut convenir à la construction d'un outil algorithmique car celui-ci devrait évoluer dans sa propre définition au fur et à mesure que la théorie se préciserait. L'autre composante est formée par la théorie algorithmique qui fixe les limites et donne les moyens de réaliser les différentes manipulations. Cette étude peut conduire à la construction d'un modèle de manipulation algorithmique plus ou moins général. Le problème devient alors une recherche d'adéquation entre une théorie linguistique particulière et le modèle algorithmique proposé. Tout traitement automatique de langues naturelles ne peut échapper à ce double aspect qui entraine le plus souvent des exigences contradictoires. Un autre aspect rarement abordé, car difficilement exprimable, rend compte de la difficulté de mise en oeuvre. Il est lié à la forme externe du langage du système employé et à ses limitations théoriques. La complexité d'adaptation d'une théorie linguistique à un modèle de manipulation est une notion essentielle dans ce type d'application (une machine de Turing travaillant sur un code binaire serait suffisante comme modèle de manipulation et sa mise en oeuvre informatique nécessiterait peu d'efforts...). Les réalisations existantes peuvent se classer en deux catégories:

- Les études qui conduisent à la réalisation d'un programme écrit soit en langage d'assemblage soit en langage évolué: Fortran, Cobol, Algol, Pascal, Pl1, Lisp, ADA, ... L'intéraction entre l'étude linguistique et l'étude algorithmique est alors maximum et une vue d'ensemble du système résultant est très difficile à atteindre. Toute évolution du système est pratiquement impossible si l'application couvre un champ assez vaste.

- Les réalisations basées sur un modèle formel. Le modèle est alors simulé sur ordinateur: Analyse hors-contexte et automate à pile Système du CETA G Veillon), Manipulation de graphes (Systeme Q A. Colmerauer, Systeme CETA J. Chauche), Automates finis généralisés (Transitions networks W. Woods), ... Dans tous les cas, l'étude algorithmique a précédé l'étude linguistique, cette dernière ayant pour but l'application du modèle proposé.

Pour analyser un système algorithmique il est nécessaire de séparer le langage défini (qui doit nécessairement exister et qui formera le méta-langage) et le modèle algorithmique sous-jacent qui impose les limites théoriques du système. Si le deuxième critère a souvent un aspect négatif par réfutation d'un modèle ayant une puissance insuffisante, le premier correspond à l' adaptabilité du système à diverses théories linguistiques. Toute théorie linguistique n'est pas adaptable à un traitement algorithmique. Les caractéristiques nécessaires sont d'ordre algorithmique et portent sur la précision de définition des règles, leur complexité (ou leur simplicité), l'importance et la forme des axiomes, l'enchainement nécessaire des règles productives, etc.... Les propriétés nécessaires à un système automatique devant servir de base à un traitement automatique de langue naturelle sont principalement la puissance du modèle sous-jacent, le langage de définition, et la complexité de mise en oeuvre. Du point de vue algorithmique, la différence entre un langage de programmation et un système de manipulation réside dans la définition du processus de manipulation et d'un ensemble de règles élémentaires. Cet aspect dégage l'utilisateur d'un ensemble de problèmes algorithmiques, par contre, il particularise les objets manipulés (bien que du point de vue théorique les modèles sous-jacents soient souvent équivalents). Un système de traitement doit donc rechercher les propriétés suivantes :

- manipuler les objets les plus complexes possibles tout en préservant une certaine efficacité.

- avoir un moyen de contrôle sur la décidabilité des algorithmes engendrés.

- avoir des moyens de mise en oeuvre simples, c'est à dire descriptifs et ne nécessitant pas une étude algorithmique pour les manipulations élémentaires.

Une part importante des théories linguistiques développées ces dernières années font apparaitre deux notions: la notion d'objet et la notion de relation entre ces objets. Le modèle naturel apparait donc être basé sur la manipulation de réseaux où les points seraient associés aux objets et les relations aux arcs. La complexité de mise en oeuvre d'un tel système conduit à particulariser la forme des réseaux

manipulables. Si le traitement s'effectue par cheminement dans le réseau, le système n'offre pas de moyen algorithmique particulier. Si, au contraire, il s'effectue par transformations successives de sous-reseaux, il permet la définition de modèles génératifs ou accepteurs. Le problème algorithmique central est, dans ce cas, la reconnaissance de sous-réseaux, la définition des choix des parties transformées, les règles d'enchainement des transformations, et enfin, les critères d'arrêt. Le système SYGMART est basé sur la transformation d'arborescences. Les objets manipulés sont plus complexes que des arborescences mais la façon de les appréhender et de les manipuler se fera par l'intermédiaire de transformation d'arborescences. Le système SYGMART est davantage un système de programmation que la simulation d'un modèle particulier. Aussi son utilisation suppose une réflexion sur la composante algortihmique de l'application que l'on veut réaliser. Mais cette réflexion doit porter seulement sur les aspects globaux de l'application, les algorithmes élémentaires nécessaires étant construits automatiquement. Le problème de la décidabilité du programme résultant est également maitrisable et permet d'obtenir des analyseurs fiables. Cet aspect est fondamental pour une application sur ordinateur. Le système SYGMART est composé de trois sous-sytèmes correspondant aux fonctions élémentaires à réaliser:

- Le système OPALE réalise le passage entre les chaînes de caractères d'entrées et les éléments structurés manipulés par le système.
- Le système TELESI réalise les manipulations des éléments structurés.
- Le système AGATE réalise le passage d'un élément structuré au résultat du système sous forme de chaîne de caractères.

Une application complète du système se résume donc à une transduction de chaîne à chaîne, le calcul de cette transduction passant par une manipulation d'éléments structurés. Des programmes auxiliaires permettent de visualiser les résultats intermediaires représentés par des éléments structurés.

La chaîne d'entrée est constituée d'une suite de caractères. Le nombre de caractères différents possibles dépend de l'ordinateur où est implanté le système. Sur une machine à octet ce nombre de caracteres dictincts est de 256, le nombre de caracteres représentables étant de 64 environ. Apres isolation, une sous-chaîne est analysée par le système OPALE. Cette sous-chaîne ne doit pas excéder 1000 caractères et correspond généralement à un mot.

Chapitre 1

Eléments manipulés par le système

1.1 Elément structuré

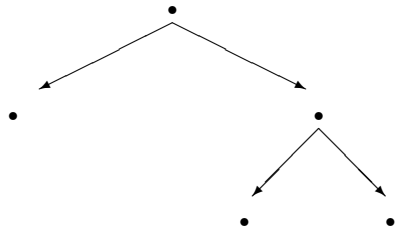
L'approche d'un élément structuré s'effectue en trois étapes:

- Arborescences.
- Arborescences étiquetées.
- Elément structuré.

1.1.1 Arborescences

Une arborescence est un graphe possédant des propriétés particulières. La définition la plus concise précise:

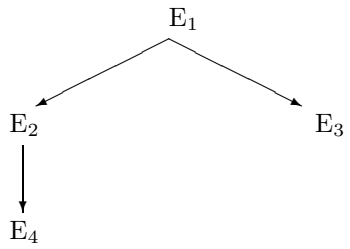
> Chaque point différent d'un point unique dénommé racine a un et un seul antécédent. La racine n'a aucun antécédent.



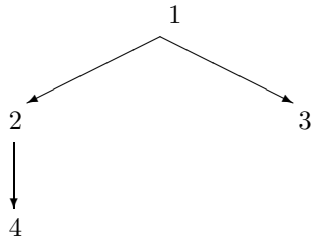
1.1.2 Arborescences étiquetées

Plusieurs approches sont possibles pour définir une arborescence étiquetée. Toutes les définitions ont pour but d'associer aux points d'une arborescence un renseignement particulier. L'ensemble des renseignements possibles constitue l'univers de référence des étiquettes. Une étiquette peut ètre simple si elle est définie par un élément unique. Elle est complexe si elle est composée d'un ensemble d'étiquettes simples. Dans ce cas, il s'agit de multi-étiquettes. Il est nécessaire de définir alors des ensembles de variables qui composeront cette multi-étiquette pour pouvoir manipuler ce type d'élément. Dans le système SYGMART chaque point n' est pas associé à une étiquette mais une fonction d'étiquetage associe à chaque point une étiquette particulière. La différence est essentielle lorsque cette fonction d'étiquetage devient non injective.

Exemple:



On peut avoir $E_2 = E_3$ mais les étiquettes E_2 et E_3 sont distinctes.



$1 \rightarrow E_1$

$2 \rightarrow E_2$

$3 \rightarrow E_2$

$4 \rightarrow E_4$

Etiquetage SYGMART. Les points 2 et 3 ont la même étiquette associée. Une modification de l'étiquette repérée par le point 2 n'entrainera pas de modification de l'étiquette repérée par le point 3 dans le premier cas, alors que dans le second, cette modification sera effective pour les deux points 2 et 3.

Une arborescence étiquetée SYGMART est donc un triplet (A,E,f) ou A est une arborescence, E un ensemble d'étiquettes, et f une application de A dans E. Une étiquette, dans ce cas, est définie par un ensemble de variables affectées de valeurs. Une étiquette peut évoluer, c'est à dire changer de valeur. On distinguera donc l'étiquette qui définit un contenant et une valeur qui définit, à un instant donné, le contenu de cette étiquette.

Trois opérations fondamentales sont définies pour le traitement des étiquettes:

- la création d'étiquette.
- la modification d'étiquette ou évolution.
- la définition ou l'évolution de la fonction d'étiquetage.

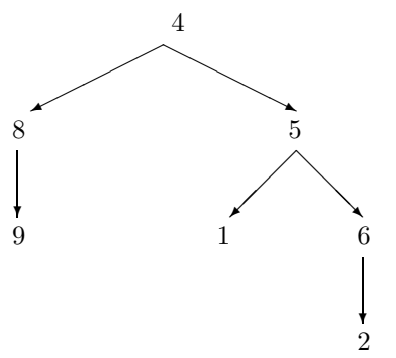
1.1.3 Elément structuré

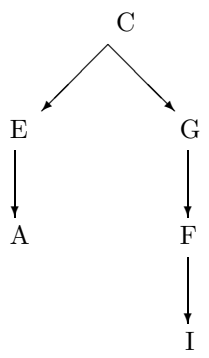
Un élément structuré est un couple (E,S) où:

- E est un ensemble fini d'étiquettes.
- S est un ensemble d'arborescences étiquetées (A,E_i,f_i) tel que $E_i \in E$.

Un élément structuré est donc un ensemble d'arborescences étiquetées sur un ensemble de points.

Exemple:





$1 \rightarrow E_1$ $\quad$ $A \rightarrow E_1$

$2 \rightarrow E_2$ $\quad$ $B \rightarrow E_2$

$C \rightarrow E_3$

$4 \rightarrow E_4$

$5 \rightarrow E_5$

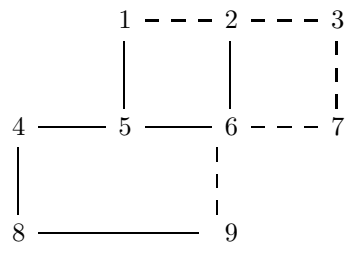
$6 \rightarrow E_6$ $\quad$ $F \rightarrow E_6$

$G \rightarrow E_7$

$8 \rightarrow E_8$

$9 \rightarrow E_9$ $\quad$ $I \rightarrow E_9$

Ce qui correspond au graphe suivant en synthétisant par rapport à l'étiquetage les deux relations arborescentes:



$1 \rightarrow E_1$
$2 \rightarrow E_2$
$3 \rightarrow E_3$
$4 \rightarrow E_4$
$5 \rightarrow E_5$
$6 \rightarrow E_6$
$7 \rightarrow E_7$
$8 \rightarrow E_8$
$9 \rightarrow E_9$

Les points de chaque arborescence sont regroupés lorsqu'ils sont associés à la même étiquette.

Dans le système SYGMART, chaque élément structuré a au plus 16 relations arborescentes. Un univers d'éléments structurés sera déterminé par la définition du nombre maximum de relations arborescentes possibles et par la description de l'ensemble des variables constituant une étiquette. L'ensemble des relations arborescentes est ordonné de 1 à 16. Le numéro d'ordre ou dimension repère une arborescence partielle de l'élément structuré. Un point d'un élément structuré n'a d'intêret pour une application qu'à travers l'étiquette qui lui est associée. Aussi, lors de la visualisation d'un élément structuré, les points ayant la mème étiquette associée seront identifiés (numérotés) de la mème manière. Cette synthèse peut s'effectuer à l'interieur d'une arborescence et (ou) sur l'ensemble des arborescences de l'élément structuré. La visualisation d'un élément par le système comprend donc deux parties: la première définit les arborescences et la deuxième la description des étiquettes. Chaque point d'une arborescence comporte deux éléments: un numéro de repère du point (numérotation canonique) et un numéro de repère d'étiquette. La visualisation de l'élément structuré précédent est la suivante:

[1] .1-1. (.2-2. (.3-3.)) , .4-4. (.5-5. , .6-6. (.7-7.)))

[2] .1-8. (.2-7. (.3-5.)) , .4-9. (.5-6. (.6-3.)))

1: E4 4: E5 7: E2
2: E8 5: E1 8: E3
3: E9 6: E6 9: E7

1.2 Déclaration des variables

La déclaration des variables permet de définir l'ensemble des variables associées à une étiquette d'un élément structuré. Cette déclaration est structurée en définition, variable globale, variable:

- Une définition définit un univers d'élément structuré. Cette définition comporte un ensemble de variables globales.
- Une variable globale détermine un bloc à l'interieur d'une définition. Ce bloc comporte un ensemble de variables.
- Une variable détermine un espace de valeurs d'un type donné.

DEFINITION: Une définition est formée d'un ensemble de variables globales et sert de référence à un univers d'éléments structurés par rapport auquel sont définis les grammaires ou les dictionnaires du système. Il peut en exister plusieurs créant ainsi plusieurs univers possibles. Le passage d'un univers à l'autre s'effectue par une modification des étiquettes: chaque étiquette garde les valeurs des variables globales communes, les autres étant éliminées ou initialisées à zéro.

VARIABLE GLOBALE: Une variable globale est donc un élément unique dans l'ensemble des déclarations et peut apparaître dans plusieurs définitions. La première apparition de cette variable doit coïncider avec sa définition. Les autres apparitions ne font que référence à cette description initiale. Une variable globale est formée d'un ensemble de variables élémentaires appartenant chacune à l'un des six types possibles: chaine, arithmetique, exclusif, non-exclusif, potentiel, référence. Il existe également un type agrégat. Ce type agrégat représente un ensemble de variables d'une même variable globale et est défini en même temps que les langages des différents systémes. L'ordre des déclarations d'une variable globale doit être impérativement celui précisé ci-dessus, un type de variable pouvant être absent. Le domaine d'évaluation des différentes variables d'une variable globale est le suivant:

type chaîne: valeur de chaîne de caractères de longueur l ($0 <= l < 1024$).

type référence: La valeur d'une variable de ce type est constituée par une référence à une autre étiquette. Lorsque cette variable est positionnée, elle réfère toujours à une étiquette même si cette dernière n'est plus associée à un point de l'élément structuré.

type arithmétique: nombre entier relatif i ($-32766 <= i <= 32765$).

type arithmétique long: nombre entier relatif i ($-2^{31} <= i <= 2^{31}$).

type flottant: nombre en virgule flottante simple précision.

type double: nombre en virgule flottante double précision.

type exclusif: variable dont la valeur peut être une des valeurs de la liste décrite après le nom de cette variable (cette liste peut être elle même une suite de variables exclusives).

type non-exclusif: variable dont la valeur peut être tout sous-ensemble de la liste des valeurs décrites après le nom de cette variable (comme pour les variables exclusives, cette liste peut être une suite de variables non exclusives).

type potentiel: variable dont la valeur peut être une des valeurs d'une liste qu'il est impossible de décrire. Cet ensemble sera alors construit au fur et à mesure qu'une valeur de cette variable sera rencontrée. On peut préciser la cardinalité maximum de l'ensemble de valeurs ainsi construit. Deux valeurs existent effectivement: Card $<= 65532$ ou Card $<= 16777216$. La valeur prise par le système sera celle immédiatement supérieure à la valeur définie dans la déclaration. Une variable potentielle aura les mêmes caractéristiques qu'une variable exclusive pour les manipulations. Deux variables potentielles peuvent avoir le même ensemble de valeurs.

La valeur nulle est définie pour toutes les variables différentes de la variable chaîne. Cette valeur correspond aussi à l'état d'une variable non affectée. Une étiquette est toujours initialisée à la valeur nulle (toutes les variables ont cette valeur nulle). Pour les variables chaînes, la valeur nulle correspond à la chaîne nulle (de longueur nulle).

Une variable globale ne peut comporter qu'une seule variable chaîne. Elle ne peut comporter qu'un nombre limité de variables élémentaires. Cette limitation correspond à une attribution de cellules de mémoires de longueur fixe. Cela correspond, par exemple, a un maximum de 16 variables potentielles courtes. Bien sûr, une définition peut comporter un nombre quasiment quelconque de variables globales.

Règles de syntaxe:

Les mots-clés de définition sont:

DEFINIT pour les définitions.

DECLARE ou DCL pour les variables globales.

FIN pour terminer une définition ou une variable globale.

REFER pour integrer dans une définition une variable globale déjà définie.

CHAINE: pour définir une variable chaîne.

REF: pour définir une ou plusieurs variables références.

ARITH: pour définir une ou plusieurs variables arithmétiques.

ARITHL: pour définir une ou plusieurs variables arithmétiques longues.

FLOTTANT: pour définir une ou plusieurs variables flottantes.

DOUBLE: pour définir une ou plusieurs variables flottantes double précision.

EXC: pour définir une ou plusieurs variables exclusives.

NEX: pour définir une ou plusieurs variables non exclusives.

POT: pour définir une ou plusieurs variables potentielles.

Ces mots clés ne doivent pas être employés comme nom de variable ou de valeur.

La structure est une structure de blocs imbriqués et le type de variable doit apparaître dans l'ordre défini ci-dessus. La structure de la définition des variables est donc la suivante:

entête: DEFINIT XXX .
entête: DECLARE YYY .
variable chaîne: CHAINE: ZZZ .
variable référence: REF: ZZZ ; ... ; ZZZ .
variable arithmétique: ARITH: ZZZ ; ... ; ZZZ .
bloc variable arithmétique longue: ARITHL: ZZZ ; ... ; ZZZ .
variable flottante: FLOTTANT: ZZZ ; ... ; ZZZ .
globale variable double précision: DOUBLE: ZZZ ; ... ; ZZZ .
variable exclusive: EXC: variable structurée;
variable structurée .
bloc variable non exclusive: NEX: variable structurée; avec les
définition ... ; variable structurée.
variable potentielle: POT: ZZZ ; ... ZZZ .
ou ZZZ(XXX− >YYY− >ZZZ')
ou ZZZ(valeur cardinale).
FIN YYY .
la séquence :
DECLARE YYY REFER XXX.
définit un bloc complet d'une variableglobale
déjà définie dans une définition précédente).
FIN XXX.

conventions suivantes:

XXX : nom de définition.

YYY : nom de variable globale.

ZZZ : nom de variable.

TTT : nom de valeur de variable structurée.

variable structurée : ZZZ (TTT , ... , TTT)
ou ZZZ (variable structurée , ... , variable structurée)

La communication de valeur entre différentes applications ayant des définitions de variables différentes s'effectue par les variables globales communes (forme 2 de l'entête).
La forme ZZZ(%XXX− >YYY− >ZZZ') d'une variable potentielle précise que les variables potentielles ZZZ et ZZZ' ont le même ensemble de valeurs de référence. Cette forme peut également être utilisée pour une variable structurée. Dans ce cas les variables ZZZ et ZZZ' doivent être de même type et auront un ensemble de valeur similaire (même nom, même ordre).

Exemple de définition de variables:

```
DEFINIT analys1.
 DECLARE varbm.
  EXC: etat(1,2,3,4,5,6,7,8,9,10,11).
   NEX: dict(1,2,3,4,5,6);
    drv(drvb(va,vn,vam,vav,vavn), drva(an,am), drvn(nm1,nm2,nm3));
    temp(pres,imp,pas,fut,cond).
   POT: nmfmt.
 FIN varbm.
FIN analys1.
DEFINIT analys2.
 DCL varbm REFER analys1.
 DCL varbs.
  CHAINE: frm.
  REF: eld;elps.
  ARITH: poida;poidb;poidc.
  NEX: cat(v,n,adjoint,deict,rep,interj,predic,coord,sub,ponct,nonlig,pref);
    sousg(sousv(vb,inf,papa,papr), sousn(ncom,npro),
    sousa(adverb,adnom,adadj,adv,adadadj,addeict,ord),
    sousd(card,artd,arti,poss,dem,rel,int,qtf,restr),
    sousr(prop,ci,rel,int,dem,poss,qtf), soussub(prep,conj)).
  POT: ul1; ul2(%analys2->varbs->ul1).
 FIN varbs.
FIN analys2.
```

Les restrictions sur la définition des noms sont les suivantes:

- Tous les noms des variables globales doivent être différents les uns des autres.
- Tous les noms de variables doivent être différents des noms des variables globales et tous les noms de variables d'une même variable globale doivent être différents les uns des autres.
- Tous les noms de valeurs d'une variable (niveau le plus élémentaire) doivent être différents les uns des autres. La valeur "0" ne peut pas être un nom de valeur.

La manipulation des variables par les différents systèmes permet de définir des affectations ou des relations. Les relations sont définies par rapport aux valeurs de variables. L' affectation comporte une variable réceptrice et la définition du calcul d'une valeur de cette variable. La définition d'une variable, réceptrice dans le cas d'une affectation, émettrice dans le cas du calcul d'une valeur, nécessite souvent une localisation de l'étiquette qui contient cette variable. Cette localisation peut prendre de multiples formes et est placée après le nom de variable. Les différentes localisations possibles sont les suivantes:

- nom: le nom est celui de l'étiquette.
- DICT(nom): étiquette obtenue par lecture d'un dictionnaire avec, comme élément d'entrée du dictionnaire, l'étiquette repérée par nom.
- DICT(nom,n): étiquette obtenue par lecture d'un dictionnaire avec, comme élément d'entrée du dictionnaire, l'étiquette repérée par nom. L'étiquette considerée sera la nième asociée à cet index.
- localisation spécifique aux sous-systèmes TELESI et AGATE :
  - ZZZ'(nom): étiquette repérée par la variable ZZZ' (qui doit être une variable référence) de l'étiquette repérée par nom.
  - DICT(ZZZ'(nom)): étiquette obtenue par lecture d'un dictionnaire avec, comme élément d'entrée du dictionnaire, l'étiquette repérée par la variable ZZZ' de l'étiquette nom.
- localisation spécifique au sous-système TELESI :
  - afct(nom1,...,nomp) : étiquette provisoire obtenue par l'application de la procédure d'affectation afct. Cette dernière procédure pouvant être avec ou sans paramètres.
  - DICT(afct(nom1,...,nomp)): étiquette obtenue par lecture d'un dictionnaire avec, comme élément d'entrée du dictionnaire, l'étiquette obtenue par la procédure d'affectation "afct". Cette dernière procédure pouvant être avec ou sans paramètres.

- ZZZ'(afct(nom1,...,nomp)) : étiquette repérée par la variable ZZZ' (qui doit être une variable référence) de l'étiquette provisoire obtenue par l'application de la procédure d'affectation afct. Cette dernière procédure pouvant être avec ou sans paramètres.
- DICT(afct(nom1,...,nomp),n): nième étiquette associée à la lecture d'un dictionnaire avec, comme élément d'entrée du dictionnaire, l'étiquette obtenue par la procédure d'affectation "afct". Cette dernière procédure pouvant être avec ou sans paramètres.
- DICT(@): Origine valable uniquement dans le sous-système TELESI. Cette origine fait référence à la dernière lecture du dictionnaire (nulle si cette dernière n'existe pas).
- DICT(ZZZ'(afct(nom1,...,nomp))): étiquette associée à la lecture d'un dictionnaire avec, comme élément d'entrée du dictionnaire, l'étiquette repérée par la variable ZZZ' (qui doit être une variable référence) de l'étiquette provisoire obtenue par l'application de la procédure d'affectation afct. Cette dernière procédure pouvant être avec ou sans paramètres.
- DICT(ZZZ'(afct(nom1,...,nomp)),n): nième étiquette associée à la lecture d'un dictionnaire avec, comme élément d'entrée du dictionnaire, l'étiquette repérée par la variable ZZZ' (qui doit être une variable référence) de l'étiquette provisoire obtenue par l'application de la procédure d'affectation afct. Cette dernière procédure pouvant être avec ou sans paramètres.

Dans le cas d'une variable chaîne, la localisation est completée par une identification éventuelle d'une sous-chaîne. Dans ce cas, la localisation comporte donc trois éléments:

- la localisation définie précédemment.
- la définition de l'origine de la sous-chaîne à extraire. Cette définition peut être:
  - un entier relatif:
    - n ou +n : nième caractère avec origine à gauche.
    - -n : nième caractère avec origine à droite.
    - variable: Nom d'une variable arithmétique, exclusive ou potentielle dont la valeur sera utilisée comme origine. Sa valeur doit être positive.

    Dans les deux cas l'origine commence à 0. C'est à dire:
    - 1er caratère à gauche: nom(0,1).
    - 1er caractère à droite: nom(-0,1).
  - un caractère:
    - 'x' : premier caractère 'x' à partir de la gauche. (sens gauche droite)
    - -'x' : premier caractère 'x' à partir de la droite. (sens droite gauche).
- la définition de la longueur de la sous-chaîne à extraire:
  - n : longueur de la sous-chaîne à extraire.
  - * : la longueur de la sous-chaîne est définie par le reste de la chaîne.
  - 'x' : la longueur de la sous-chaîne est définie par le premier caractère 'x' rencontré à partir de l'origine et dans le sens du parcours défini par cette origine (+: sens gauche droite, -: sens droite gauche).
  - variable: Nom d'une variable arithmétique, exclusive ou potentielle dont la valeur sera utilisée comme longueur. Sa aleur doit être positive.

Un nom particulier de localisation d'étiquette est le nom de format. Un format est une étiquette fixe dont les valeurs sont celles définies par la dernière affectation. Un format peut donc être utilisé comme un élément constant s'il n'est jamais affecté en dehors de sa définition. Il peut également servir de position constante ou registre. Dans tous les systèmes un format peut-être lu ou modifié. La modification d'un format doit appartenir à une partie affectation quelconque. La partie lecture peut appartenir à un traitement quelconque (condition ou affectation).

Pour chaque système, les types de localisations possibles seront précisés. La localisation peut être implicite et, dans ce cas, ne doit pas être mentionnée. Lorsque des variables exclusives, non execlusive, ou potentielles sont de même dimension interne, elles peuvent être appliquées les unes sur les autres. Dans

ce cas, cela correspond à la définition d'une application d'un ensemble de valeurs dans un autre ensemble de valeurs. Cette application est identifiée comme un décalage de valeur et la variable correspondante une variable décalée. Pour définir ce décalage il est nécessaire de préciser la variable réceptrice et la variable émettrice. La forme syntaxique est la suivante:

%(ZZZ)< −ZZZ' ZZZ est la variable réceptrice, ZZZ' la variable émettrice.

Du fait que deux variables peuvent porter le même nom si elles n'appartiennent pas à la même variable globale, il est nécessaire de pouvoir préciser une variable à partir du seul élément unique, le nom de sa variable globale. On a alors un nom de variable repéré: ZZZ− >ZZZ'− >ZZZ" : la variable ZZZ" fait partie de la variable ZZZ' qui fait elle-même partie de la variable ZZZ. Cette construction peut également repérer un élément d'une variable structurée, ou une valeur d'une variable dont le nom est identique à un nom de variable.

Enfin une valeur de variable ou constante est définie par la nature de la variable de référence de cette constante.

Les differents types possibles sont donc les suivants:

- Variable simple: ZZZ
- Variable repérée: ZZZ− >ZZZ'
- Variable localisée: ZZZ(nom)
- Variable repérée localisée: ZZZ− >ZZZ'(nom)
- Variable référencée localisée: ZZZ(ZZZ'(nom))
- Variable repérée référencée localisée: ZZZ− >ZZZ'(ZZZ"(nom))
- Variable dictionnaire: ZZZ(DICT(nom))
- Variable dictionnaire indexée: ZZZ(DICT(nom,n))
- Variable dictionnaire repérée: ZZZ− >ZZZ'(DICT(nom))
- Variable dictionnaire repérée indexée: ZZZ− >ZZZ'(DICT(nom),n)
- Variable dictionnaire référencée: ZZZ(DICT(ZZZ'(nom)))
- Variable dictionnaire référencée indexée: ZZZ(DICT(ZZZ'(nom),n))
- Variable dictionnaire repérée référencée: ZZZ− >ZZZ'(DICT(ZZZ"(nom)))
- Variable dictionnaire repérée référencée indexée: ZZZ− >ZZZ'(DICT(ZZZ"(nom),n))
- Variable dictionnaire calculée: ZZZ(DICT(afctd(nom1, nom2, nom3)))
- Variable dictionnaire calculée indexée: ZZZ(DICT(afctd(nom1, nom2, nom3),n))
- Variable dictionnaire repérée calculée: ZZZ− >ZZZ'(DICT(afctd(nom1,nom2)))
- Variable dictionnaire repérée calculée indexée: ZZZ− >ZZZ'(DICT(afctd(nom1,nom2),n))
- Variable decalée localisée: %(ZZZ)< −ZZZ'(nom)
- Variable repérée decalée localisée: %(ZZZ− >ZZZ')< −ZZZ"− >ZZZ"'(nom)
- Variable decalée référencée localisée: %(ZZZ)< −ZZZ'(ZZZ"(nom))
- Variable dictionnaire decalée: %(ZZZ)< −ZZZ'(DICT(nom))
- Variable dictionnaire decalée indexée: %(ZZZ)< −ZZZ'(DICT(nom,n))
- Variable dictionnaire decalée référencée: %(ZZZ)< −ZZZ'(DICT(ZZZ"(nom)))

- Variable dictionnaire decalée référencée indexée: %(ZZZ)< −ZZZ'(DICT(ZZZ"(nom),n))
- Variable dictionnaire decalée calculée: %(ZZZ)< −ZZZ'(DICT(afctd(nom1,nom2)))
- Variable dictionnaire decalée calculée indexée: %(ZZZ)< −ZZZ'(DICT(afctd(nom1,nom2),n))
- Variable chaîne simple: ZZZ , ZZZ(0,3) , ZZZ(-2,'a')
- Variable chaîne repérée: ZZZ− >ZZZ' , ZZZ− >ZZZ'(2,4)
- Variable chaîne localisée: ZZZ(nom) , ZZZ(nom,2,4)
- Variable chaîne repérée localisée: ZZZ− >ZZZ'(nom), ZZZ− >ZZZ'(nom,0,1)
- Variable chaîne référencée: ZZZ(ZZZ'(nom)), ZZZ(ZZZ'(nom),-'a','b')
- Variable chaîne repérée référencée: ZZZ− >ZZZ'(ZZZ"(nom)), ZZZ− >ZZZ'(ZZZ"(nom),0,1)
- Variable chaîne dictionnaire: ZZZ(DICT(nom)), ZZZ(DICT(nom),2,3)
- Variable chaîne dictionnaire indexée: ZZZ(DICT(nom)), ZZZ(DICT(nom,n),2,3)
- Variable chaîne dictionnaire repérée: ZZZ− >ZZZ'(DICT(nom)), ZZZ− >ZZZ'(DICT(nom),2,3)
- Variable chaîne dictionnaire repérée indexée: ZZZ− >ZZZ'(DICT(nom,n)),
  ZZZ− >ZZZ'(DICT(nom,n),2,3)
- Variable chaîne dictionnaire référencée: ZZZ(DICT(ZZZ'(nom))), ZZZ(DICT(ZZZ'(nom)),3,6)
- Variable chaîne dictionnaire référencée indexée: ZZZ(DICT(ZZZ'(nom),n)),
  ZZZ(DICT(ZZZ'(nom),n),3,6)
- Variable chaîne dictionnaire repérée référencée: ZZZ− >ZZZ'(DICT(ZZZ"(nom))),
  ZZZ− >ZZZ'(DICT(ZZZ"(nom)),2,6)
- Variable chaîne dictionnaire repérée référencée indexée: ZZZ− >ZZZ'(DICT(ZZZ"(nom),n)),
  ZZZ− >ZZZ'(DICT(ZZZ"(nom),n),2,6)
- Variable chaîne dictionnaire calculée: ZZZ(DICT(afctd)), ZZZ(DICT(afctd(nom1,nom2)),4,5)
- Variable chaîne dictionnaire calculée indexée: ZZZ(DICT(afctd),n),
  ZZZ(DICT(afctd(nom1,nom2),n),4,5)
- Variable chaîne dictionnaire repérée calculée: ZZZ− >ZZZ'(DICT(afctd(nom1,nom2))),
  ZZZ− >ZZZ'(DICT(afctd),2,3)
- Variable chaîne dictionnaire repérée calculée indexée: ZZZ− >ZZZ'(DICT(afctd(nom1,nom2),n)),
  ZZZ− >ZZZ'(DICT(afctd,n),2,3)

Une constante correspond à une valeur de variable. Elle est donc identifiée par un nom ou une valeur. Le nom correspond à celui défini dans les déclarations. Les valeurs sont placées entre apostrophes sauf pour les valeurs de variable arithmétique (variable entière). Si cette valeur doit contenir une apostrophe cette dernière doit être doublée ou remplacée par le caractère spécial "\'". La valeur nulle ("0") est définie pour toutes les variables et correspond à une variable non affectée à l'exception des variables de type référence. Dans le cas d'une variable de type référence "0" peut être un nom de point et dans ce cas ce nom a priorité sur la valeur nulle. Les différentes possibilités de définition d'une constante sont les suivantes:

- Nom: 1 va adjoint
- Nom de variable potentielle: 'valeur' 'uldf'
- Valeur arithmétique: 24 45 -5 +8 0
- Valeur flottante: '3.14' '-4.25' '12.8e-3' '5'
- Valeur de variable référence: nom (d'une etiquette)
- Chaîne: 'chaîne' 'd"une' 'jusqu\'à' (avec les caractères spéciaux: '\\' '\n' '\t' \r' '\f')
- Valeur repérée: ZZZ− >val ul1− >'unité' sousg− >sousa− >adv

1.3 Opérations sur les variables

1.3.1 Expression

Une expression permet de définir une valeur de variable. Elle peut être définie par une constante ou une variable localisée ou une expression parenthésée construite avec des opérateurs qui dépendent du type de variable de l'expression.

opérateur et fonctions pour les variables de type chaîne:

- || concaténation.
- %CHAINE chaîne correspondant au nom d'une valeur de variable potentielle, exclusive ou non exclusive.
- %DBLCH conversion flottant double precision → chaîne.
- %FLTCH conversion flottant → chaîne.
- %ENTCH conversion variable arithmétique → chaîne.
- %IMGB définition d'une chaîne image binaire de la valeur d'une variable. Cette chaîne n'est pas toujours imprimable.
- %ESPACE chaîne d'espaces de longueur définie par une constante entière.
- %NOMFICH nom du fichier d'entrée traité.
- %TEXT chaîne correspondant à une partie du texte d'entrée. Cette fonction à deux paramètres: deux variables arithmétiques, la première représentant l'origine (en octets) de la chaîne à extraire, le second sa longueur .
- %TCHAINE chaîne résultant d'une substitution. Cette fonction à trois paramètres: trois définitions de chaînes: la première correspond à la chaîne source, le second la chaîne à substituée et le troisième la chaîne de substitution. Le résultat correspond à la première chaîne ou la substitution à été effectuée. Le second paramètre peut être précédé d'un caractère '+', '-' ou '*'. '+' signifie que seule la première occurence dans le parcours droite gauche doit être substituée; '-' la première occurence dans le parcours gauche droite; et '*' toutes les occurences. L'abscence de l'un de ces caractères correspond à '+'.
- %TRANSCHAINE translation d'une chaîne. Cette fonction à trois paramètres: trois définitions de chaînes: la première correspond à la chaîne source, le seconde la chaîne de l'ensemble des caractères à translater et la troisième l'ensemble des caractères images. Le résultat correspond à la première chaîne où la substitution de chaque caractère appartenant à la deuxième chaîne a été remplacé par le caractère de même rang de la troisième chaîne. Si la troisième chaîne est plus courte que la seconde certains caractère n'auront pas d'image et seront donc effacés.

opérateurs et fonctions pour les variables de type arithmétique:

- + somme.
- - différence.
- * produit.
- :- division entière.
- %PROF: profondeur d'un point dans l'arborescence.
- %NBDIM: nombre de dimensions traitées.
- %ADRQ: adresse d'une étiquette.
- %LGCH: longeur d'une chaîne.
- %PLFICH: position d'un mot (en nombre de caractères) dans le fichier d'entrée.
- %LGTEXT: nombre de caractères total du fichier d'entrée. (Lorsque cette fonction est appelée par le système OPALE la valeur obtenue correspond au nombre de caractères lus à cet instant).
- %CHENT: conversion chaîne → entier.

- %IVIMEN: inverse image binaire → entier.
- %IVIMEL: inverse image binaire → entier long.
- %FLTENT: conversion flottant → entier.
- %DBLENT: conversion flottant double précision → entier.

Les fonctions "%LGCH" et "%CHAINE" ne s'appliquent que sur des variables simples, repérées ou localisées.
La fonction "%LGCH" a un paramètre optionnel supplémentaire : M ou L qui indique le mode de calcul (cf "%LGTEXT").
La fonction "%ESPACE" n'est valable que pour le dictionnaire des formats, la fonction "%PLFICH" que pour le sous-système OPALE et les fonctions "%PROF", "%NBDIM" et "%ADRQ" que pour le sous-système TELESI.
La fonction "%ESPACE" a comme parmètre une constante entière.
Les fonctions "%PLFICH" et "%LGTEXT", ont comme paramètre optionnel soit le symbole M (matriel) soit le symbole L (Logiciel). Ce symbole indique si la valeur doit être calculée en octets (M) ou en caractères (L), un caractère pouvant occuper plusieurs octets suivant le codage. L'absence de paramètre est équivalent à M. Le calcul logiciel est dynamique et dépend du fichier des messages (cf MISE EN OEUVRE). Ce calcul comprend soit un codage fixe (nombre d'octets constants par caractère) soit un codage variable qui est alors utf8, utf16 ou utf32.
Les fonctions "%NBDIM" et "%NOMFICH" n'ont pas de paramètre.
La fonction "%TEXT" a deux paramètres qui doivent être des variables arithmétiques localisées. La premi'ere indique la position du début de la chaîne à extraire en nombre d'octets depuis le début du texte. La seconde détermine la longueur de la chaîne à extraire en nombre d'octets .
Pour les fonctions %TCHAINE et %TRANSCHAINE le premier paramètre ne peut pas être une constante si cette fonction se trouve en partie gauche d'une relation. Les fonctions spécifiques au sous-système TELESI ont comme paramètre le nom d'un point d'un schéma (éventuellement remplacé par "*" pour les conditions propres).

opérateurs et fonctions pour les variables de type flottant:

- + somme.
- - différence.
- * produit.
- :- division.
- %CEIL: fonction seuil.
- %EXP: fonction exponentielle
- %FABS: fonction valeur absolue.
- %FLOOR: fonction palier.
- %LOG: fonction logarithme népérien.
- %LOG10: fonction logarithme décimal.
- %SQRT: fonction racine carrée.
- %ACOS: fonction arc-cosinus.
- %ASIN: fonction arc-sinus.
- %ATAN: fonction arc-tangente.
- %COS: fonction cosinus.
- %COSH: fonction cosinus hyperbolique.
- %SIN: fonction sinus.
- %SINH: fonction sinus hyperbolique.
- %TAN: fonction tangente.
- %TANH: fonction tangente hyperbolique.
- %J0: fonction de Bessel.
- %J1: fonction de Bessel.

Toutes ces fonctions sont issues de la bibliothèque mathématique standard des compilateurs C. Elles n'ont qu'un seul paramètre qui doit être de type entier, flottant ou double precision. Elle produisent un résultat qui est également en double précision. Si le paramètre n'est pas en double précision il est alors converti avant l'appel de la fonction. Si le nom de la fonction est suivi de "[FL]" ou "[ENT]" ce résultat sera converti en flottant ou en entier.

- %CHFLT: conversion chaîne → flottant.
- %CHDBL: conversion chaîne → flottant double précision.
- %IVIMFL: conversion inverse image binaire → flottant.
- %IVIMDB: conversion inverse image binaire → flottant double précision.
- %FLTDBL: conversion flottant → flottant double précision.
- %DBLFLT: conversion flottant double précision → flottant.
- %ENTFLT: conversion entier → flottant.
- %ENTDBL: conversion entier → flottant double précision.

opérateurs pour les variables non exclusives:

- | union.
- & intersection.
- ^ complément.

fonction pour les variables de type référence:

- %ETIQUETTE: création ou référence à une étiquette définie par une valeur de chaîne. Cette fonction est définie seulement dans le sous-système TELESI, a comme paramètre une variable de type chaîne et retourne une valeur référence. Lorsque plusieurs appels sont effectués avec la même valeur de chaîne les valeurs retournées sont identiques. Cette référence ne correspond qu'à une abréviation, c'est à dire qu'elle ne peut être construite que dans une affectation définie lors d'une transformation et que sa référence n'est valable que simultanément aux autres étiquettes (elle peut disparaître lors d'un retour arrière).

1.3.2 Relations

Les relations permettent de définir des conditions d'applications dans les différents systèmes. Une relation peut être composée d'un ensemble de relations élémentaires reliées entre elles par des connecteurs. Dans ce cas, chaque relation élémentaire doit être placée entre parenthèses et les connecteurs possibles sont:

- | connecteur ou
- & connecteur et
- ! connecteur non.

Les relateurs permettent de définir des relations élémentaires et dépendent du type des variables en présence. On distingue trois familles de relateurs, les relateurs opérant sur tous les types de variables, ceux opérant sur les variables non exclusives et ceux opérant sur les variables arithmétiques.

relateurs opérant sur tous les types:

- = égalité
- ! = différence

relateurs opérant sur les variables non exclusives:

Les variables non exclusives sont des variables ensemblistes. Les relateurs seront donc des relateurs ensemblistes. Deux séries de relateurs sont définies car souvent il est nécessaire d'adjoindre à une relation la relation non vide. Si, par exemple, on définit la relation inclusion, on aura la relation a inclus dans b, et la relation a non vide et inclus dans b.

relateurs classiques:

- $ >= inclusion large
- $ > inclusion stricte
- !$ >= non inclusion large
- !$ > non inclusion stricte
- $ <= appartenance large
- $ < appartenance stricte
- !$ <= non appartenance large
- !$ < non appartenance stricte

relateur impliquant une condition non vide: Tous ces opérateurs sont de la forme (A op B) et (X ! = 0). L'opérateur "op" correspond à un des opérateurs précédent et X à une opérande A ou B.

- @ >= inclusion large (X = B)
- @ > inclusion stricte (X = B).
- !@ >= non inclusion large (X = A).
- !@ > non inclusion stricte (X = A).
- @ <= appartenance large (X = A).
- @ < appartenance stricte (X = A).
- !@ <= non appartenance large (X = B).
- !@ < non appartenance stricte (X = B).

relateur opérant sur les variables arithmétiques:

- < inférieur
- > supérieur
- <= inférieur ou égal
- >= supérieur ou égal
- ! <= non inférieur ou égal
- ! >= non supérieur ou égal
- ! < non inférieur
- ! > non supérieur

La fonction "%EGALSTR" definit une relation entre deux points d'un schéma. Elle est vraie si les arborescences dont ces parametres constituent les racines sont égales. Cette fonction n'est définie que pour les conditions inter-sommets d'un schéma TELESI.

%EGALSTR(P1,P2) est vraie si les arborescences de racines P1 et P2 sont égales.

Deux arborescences sont égales si les structures sont égales et si deux points correspondants de ces structures font références à la même étiquette.

Remarque :
La parenthèse ouvrante qui suit le nom d'une fonction ne doit être précédée d'aucun blanc.

exemple %TAN(XXX) pour obtenir la tangente.

1.3.3 Affectations

Les affectations permettent de modifier les valeurs des différentes variables d'une étiquette. Elles sont toutes définies suivant la même syntaxe:

variable à affecter = expression

Le système TELESI comporte en plus des affectations conditionnelles. Ces affectations ont la structure de "si" imbriqués. La forme générique est la suivante:

si condition alors liste d'affectations sinon liste d'affectations .

la syntaxe est la suivante:

< condition : liste d'affectations # liste d'affectations >

la liste d'affectations est une suite d'affectations conditionnelles ou non séparées par des points virgules. la première liste correspond à la condition vraie, la seconde à la condition fausse.

1.4 Identification et référence

1.4.1 Noms externes

Tous les langages des grammaires et dictionnaires font référence à une définition de variables. Cette définition fait implicitement référence à un univers d'éléments structurés. Le dictionnaire d'étiquettes (ou dictionnaire de formats) est unique dans le système et comporte un ensemble d'étiquettes pour chaque définition. Cet ensemble d'étiquettes est précédé de la définition de référence.

syntaxe: &REFER(XXX).

où XXX est le nom de la définition.

Tous les autres éléments (langages OPALE, TELESI, AGATE, dictionnaire de chaînes, dictionnaire d'étiquettes) sont précédés de la définition de référence et d'un identifiant. Cet identificateur est un nom externe et doit être unique pour chaque grammaire ou dictionnaire (bien sûr une grammaire OPALE et une grammaire TELESI par exemple peuvent avoir le même nom mais deux grammaires OPALE ou deux grammaires TELESI doivent avoir des noms distincts). Ce nom sera utilisé pour identifier la grammaire ou le dictionnaire au cours de l'éxecution du système compilé sur un texte.

syntaxe: &REFER(XXX,NNN).

où XXX est le nom de la définition et NNN le nom externe identifiant l'élément.

1.4.2 Variable agrégat

Plusieurs variables d'une même variable globale peuvent être regroupées pour le traitement. Cet ensemble de variables sera considéré comme une variable non exclusive et ne devra pas comprendre de variable de type chaîne. l'intérêt de ce type de variable réside dans le gain de temps d'éxecution qu'il entraine ainsi que dans la simplicité d'écriture des éléments des differents langages (règles ou éléments de dictionnaires). Un ensemble de variables d'une même variable globale ainsi regroupé définit une variable agrégat identifiée par un nom. Ce nom doit être différent des noms des variables globales de la définition concernée ainsi que des noms de variables de la variable globale ou cette variable agrégat est définie. Cette variable agrégat est une variable locale dans le sens où elle n'est définie que pour l'élément où elle apparait (grammaire ou dictionnaire). La définition des variables agrégats suit immédiatement la définition de référence de la grammaire ou du dictionnaire l'utilisant. Une définition de variable agrégat est composée du nom de cette variable agrégat, du nom de la variable globale où seront extraites les variables composantes, et de la liste des variables composantes qui doivent toutes appartenir à la variable globale précisée et ne pas être de type chaîne.

syntaxe: VVV = YYY(ZZZ1, . . . , ZZZn).

où VVV est le nom de la variable agrégat a définir,
YYY le nom de la variable globale concernée,
ZZZ1 ... ZZZn la liste des variables composantes.

L'emploi des variables agrégats permet de réduire l'écriture des différents traitements d'étiquettes et d'améliorer également le temps d'exécution de ce traitement.

1.4.3 Variable externe

Une variable chaîne peut être interprétée comme un tableau de variables entières ou flottantes. Le mot clé, qui remplace le mot "CHAINE" dans la définition des variables est "VARIABLE:". Une variable globale ne peut comporter qu'une seule variable externe et, dans ce cas, cette variable globale ne peut pas comporter de variable de type chaîne. Sa déclaration prend alors la place d'une variable de type chaîne. Cette variable est traitée de manière spécifique et sa valeur n'est jamais visualisée. Le paramètre "longueur" d'une variable chaîne doit être d'efini par le mot clé du type de variable concernée qui peut être ARITH, ARITHL, FLOTTANT ou DOUBLE. L'index du premier élément d'un vecteur est 0. La définition de l'étiquette contenant le vecteur peut être :

- un point d'un schéma,
- un format,

- une valeur de variable référence d'un point d'un schéma,
- une valeur de variable référence d'un format,
- une origine issue de la lecture d'un dictionnaire avec comme entrée une étiquette associée à un point du schéma,
- une origine issue de la lecture d'un dictionnaire avec comme entrée une étiquette une étiquette définie par une variable référence d'une étiquette associ'ee à un point d'un schéma,
- une étiquette résultat de l'application d'une procédure d'affectation,
- une origine issue de la lecture d'un dictionnaire avec comme entrée l'étiquette résultant d'une procédure d'affectation.

L'indice peut être :

- une constante,
- une variable arithmétique appartenant à une étiquette associée à un point d'un schéma,
- une variable arithmétique appartenant à un format.

  exemple:

  FRM(EC,3,4) : variable chaîne.
  FRM(EC,3,ARITH) : quatrième élément d'un vecteur d'entiers.

Un ensemble de fonctions permet le traitement de ces vecteurs:

- %LGVECT: Nombre d'éléments d'un vecteur. La valeur de cette fonction est arithmétique et il faut préciser le type des éléments du vecteur:

  %LGVECT(<Nom de variable externe>,<type>)

- %VECTEUR: construction d'un vecteur. Le résultat de cette fonction est défini par un vecteur de type définit par le premier paramètre. Cette fonction comprend quatre formes.
  - Dans la première forme le nombre d'éléments est égal au nombre de paramètres. Le mot clé "INDICES:" est équivalent à "ARITH:". Il est défini pour précisé l'usage souhaité du vecteur. Avec ce mot clé les valeurs ne pourront pas être supérieure à 4096. Les paramètres suivants sont des valeurs de cette variable et constituent les valeurs entières des éléments de ce vecteur.

    vecteur = %VECTEUR(ARITH:1,2,6,4,5)
    vecteur = %VECTEUR(INDICES:1,2,6,4,5)
    vecteur = %VECTEUR(UL:'cas1','cas2','cas3')

    La valeur d'un indice issue d'une variable exclusive ou potentielle est toujours supérieur à 0. Le nombre associé correpond au rang d'apparition de la valeur de cette variable. Pour maitriser cette valeur il est toujours possible de définir un ensemble de valeurs initiales d'une variable potentielle.
  - Dans la deuxième forme la longueur est fixés par le premier paramètre qui est un entier. Le type est déterminé par le troisième paramètre qui est d'un mot clé de type: 'ARITH', 'ARITHL','FLOTTANT' ou 'DOUBLE'.

    Le deuxième paramètre peut être:
    - une constante
    - une valeur de variable
    - La forme "%P" associant un vecteur indice.

    Dans ces trois cas la forme et la signification sont identique à celle du deuxième opérande de la fonction "OPERATIONV".

    vecteur = %VECTEUR(256,4,ARITH)
    vecteur = %VECTEUR(256,VAL(4),ARITH)
    vecteur = %VECTEUR(256,%P(VectIndice,1),ARITH)

– Dans la troisième forme le vecteur est construit à partir d'un vecteur existant. Le nouveau vecteur sera semblable à l'ancien mais certaines composantes seront annulées. Trois cas sont possible:

  * Seules les plus fortes valeurs seront conservées :
    %VECTEUR(SUP[variable arithmétique]:vecteur,TYPE)
  * Seules les plus faibles valeurs seront conservées :
    %VECTEUR(INF[variable arithmétique]:vecteur,TYPE)
  * Seules les valeurs supérieures au seuil seront conservées :
    %VECTEUR(SEUIL[variable seuil]:vecteur,TYPE)

  La variable arithmétique contient le nombre d'éléments concernés.

– La quatrième forme concerne les traitements maticiels: multiplication matricielle et combinaison linéaire.

  1. La multiplication matricielle est identifiée par le mot "MAT:". Dans cette forme le vecteur est le résultat d'une multiplication matricielle et d'un vecteur existant. Le nouveau vecteur aura comme nombre de composantes le nombre de ligne de la matrice. La matrice sera définie dans le dictionnaire des formats. L'opération s'effectue éventuellement avec des conversions.

     vecteur = %VECTEUR(MAT:<Nom Matrice>,vecteur,TYPE)

  2. Les combinaisons linéaires sont définie par le mot "LIN:". Le nouveau vecteur sera défini par une combinaison linéaire de toutes les lignes de la matrice. Chaque ligne sera affectée d'un coéfficient égal à la valeur de la composante du vecteur paramètre. Ansi si "matr" et "vect" sont les noms de la matrice et du vecteur, le vecteur résultat sera :

     $\sum matr[i] * vect[i]$ où $matr[i]$ correspond au vecteur ligne.

     L'opération s'effectue éventuellement avec des conversions. L'ajustement des dimensions est la suivante: La dimension du vecteur résultant est celle des lignes. Si le vecteur paramètre a une dimension plus grande que le nombre de lignes, les valeurs exédentaires sont ignorées. Si, par contre, le vecteur paramètre a une dimension inférieure au nombre de lignes, les coéfficients complémentaires sont nuls. La forme syntaxique est semblable à la précédente :

     vecteur = %VECTEUR(LIN:<Nom Matrice>,vecteur,TYPE)

- %NORMVECT: Cette fonction a comme valeur la norme d'un vecteur. Cette valeur est de type "flottant double précision".

- %SOMMEVECT: Cette fonction a comme valeur la somme de tous les éléments d'un vecteur. Cette valeur est de type flottant double précision.

  Ces deux fonctions ont deux paramètres: un vecteur et un type.

- %MAXVECT: Cette fonction a comme valeur la valeur maximum des composantes du vecteur.

- %MINVECT: Cette fonction a comme valeur la valeur minimum des composantes du vecteur.

  Pour ces deux fonction le résultat est du type flottant double précision.

  Pour les quatre fonctions précédentes, il est possible de définir une conversion: le nom de la fonction doit être suivit de "[FL]" ou "[ENT]".

- %FVECTEUR: fonctions vectorielles dont le résultat est une variable flottante double précision:

  syntaxe: %FVECTEUR(<opération>,<vecteur 1>,<vecteur 2>,<type>)

  L'opération peut être:

  – DIST : distance euclidienne <vecteur 1>, <vecteur 2>.
  – SCAL : produit scalaire <vecteur 1>, <vecteur 2>.

Le résutat de ces fonctions peut être converti si le nom est suivi de "[FL]" ou "[ENT]".

exemple: comparaison de valeur (distances ou sommes):

%FVECTEUR(DIST,vect1,vect2,DOUBLE) < %FVECT(DIST,vect1,vect3,DOUBLE)
F = %FVECTEUR[FL](DIST,vect1,vect2,DOUBLE);
%SOMMEVECT(vect1,ARITH) = %SOMMEVECT(vect2,FLOTTANT)

Dans ce dernier cas la comparaison est possible car la valeur de la fonction somme est toujours exprimée en double précision. Pour avoir une somme entière il est nécessaire d'effectuer une conversion:

I = %SOMMEVECT[ENT](vect1,ARTITH).

- %OPERATIONV: fonctions vectorielles:

Cette fonction n'a pas de valeur. Elle correspond à un appel de procédure et constitue donc une affectation en soi. Cette fonction prend quatre formes. La première forme correspond à une opération binaire, la deuxième à une opération de conversion, la troisième à une opération de permutation, et la quatrième à une opération dont la table est définie par une matrice.

Première forme:

%OPERATIONV(<opération>,<vecteur 1>,<vecteur 2>,<vecteur 3>,<type>)

- + : somme : $x_{i_1} = x_{i_2} + x_{i_3}$.
- - : difference $x_{i_1} = x_{i_2} - x_{i_3}$.
- * : produit $x_{i_1} = x_{i_2} * x_{i_3}$.
- / : quotien $x_{i_1} = x_{i_2}/x_{i_3}$.

Deuxième forme:

%OPERATIONV(%,<vecteur 1>,<type 1>,<vecteur 2>,<type 2>)

Le "<vecteur 1>" est affecté avec le vecteur issu du "<vecteur 2>" et dans lequel chaque élément a été converti du "<type 2>" vers le "<type 1>".

Troisième forme:

%OPERATIONV(~,<vecteur 1>,<vecteur 2>,<vecteur 3>,<type>)

Le "<vecteur 1>" est affecté avec le vecteur ayant le même nombre d'élements que le "<vecteur 2>" et est du type <type>. Le <vecteur 3> est également du type <type>. Le <vecteur 2> est un vecteur indice qui indique pour chaque valeur d'index l'élément du <vecteur 3> à affecté. Ansi V1[i] est affecté à la valeur V3[V2[i]]. Une valeur de V2[i] qui est spérieure au nombre déléments de V2 est ignorée. Si un indice est absent, la valeur correspondante de V1 est mise à 0.

Quatrième forme:

%OPERATIONV(<Nom de matrice>,<vecteur 1>,<vecteur 2>,<vecteur 3>,ARITH)

Chaque composante du vecteur 1 est obtenue à partir d'une opération dont la table est définie par la matrice.
Ansi V1[i] est affecté avec la valeur Mat[V2[i]][V3[i]]. Si les valeurs des opérandes dépassent les dimensions de la matrice la valeur affectée est nulle. Le vecteur 2 peut aussi être une matrice. Dans ce cas le vecteur 3 est un vecteur indice et le vecteur 1 est affecté du résultat de l'opération définie par la matrice Mat appliquée sur tous les vecteurs lignes de la deuxième matrice dont les indices sont définis dans le vecteur 3. Ainsi V1[i] est affecté avec la valeur $\bigotimes_{j \in V3} Mat2[j][i]$, l'opération "$\bigotimes$" étant définie par la matrice Mat.

Le vecteur résultat a toujours le même nombre d'éléments que celui de l'opérande qui en a le moins. Il y a donc troncature éventuelle d'une des opérandes.

Dans le cas des fonctions "FVECTEUR" et "OPERATIONV" (1ère forme) les opérandes qui sont des vecteurs peuvent être définies de plusieurs façons:

- Le premier opérande peut être un vecteur nul et dans ce cas la forme "%N(<nombre>)" remplace le nom de vecteur et "nombre" défini la dimension du vecteur.
- Le deuxième opérande peut être
  - ∗ une constante ou une variable de même type que celui défini en dernier paramètre. Cette forme correspond à un vecteur dont tous les éléments sont égaux à cette valeur. La dimension du vecteur est alors celle du premier opérande.
  - ∗ Un vecteur obtenu à partir d'un vecteur considéré comme vecteur indice et sa dimension est celle du premier opérande. Ce vecteur est nul pour tout indice différent d'un indice défini dans le vecteur indice. Les valeurs non nulles sont toutes égales soit
    - · à une constante,
    - · à une valeur de variable.

  Pour obtenir ce vecteur il faut utiliser la forme "%P(<vecteur>,<valeur>)" à la place du deuxième opérande.
  La valeur peut être multipliée par une valeur d'un autre vecteur indice. Dans ce cas une valeur non nulle est obtenue par le produit de la constante ou de la variable avec la valeur de l'indice de même rang du deuxième vecteur indice. La forme syntaxique utilise alors la fonction "%V":

  %P(<vecteur>,%V(<vecteur>,<valeur>))

  Lorsque plusieurs indices sont identiques dans le vecteur indice la valeur du vecteur résultant pour cet indice correspond à la somme des valeurs obtenues pour chaque indice identique.

Dans tous les cas les vecteurs définissant tout ou partie d'une opérande peuvent être obtenus par la lecture d'un dictionnaire.

exemple:

multiplication d'un vecteur par une constante:

```
%OPERATIONV(*,resultat,vecteur,constante,ARITH).
```

addition sur des composantes dont l'indices est défini:

```
%OPERATIONV(+,resultat,vecteur,%P(vect_indice,1),FLOTTANT).
```

addition de deux vecteurs issus de lectures de dictionnaires.

- Le premier est un vecteur simple de flottants associé à une entrée du dictionnaire.
- Le deuxième est un vecteur construit à partir de deux vecteurs indices issus de la lecture du dictionaire:
  l'un (le premier) donne les composantes non nulle.
  l'autre (le deuxième) définit les valeurs à affecter.

```
%OPERATIONV(+,resultat,vecteur(DICT(1)),
  %P(vect_indice(DICT(2)),%V(vect_mult(DICT(2,2)),'.1')),FLOTTANT).
```

Pour les variables externes la fonction %CHAINE permet de convertir le tableau en chaîne de caractères. Il est alors nécessaire de préciser le type:

%CHAINE(<Nom de variable externe>,<type>)

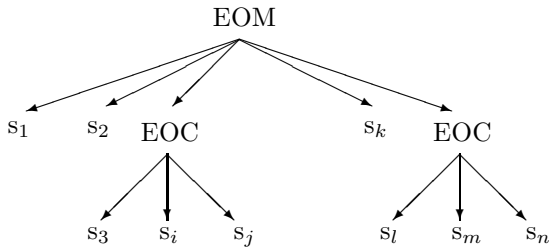
Dans la conversion, chaque élément est séparé par une virgule.

Chapitre 2

Le langage OPALE

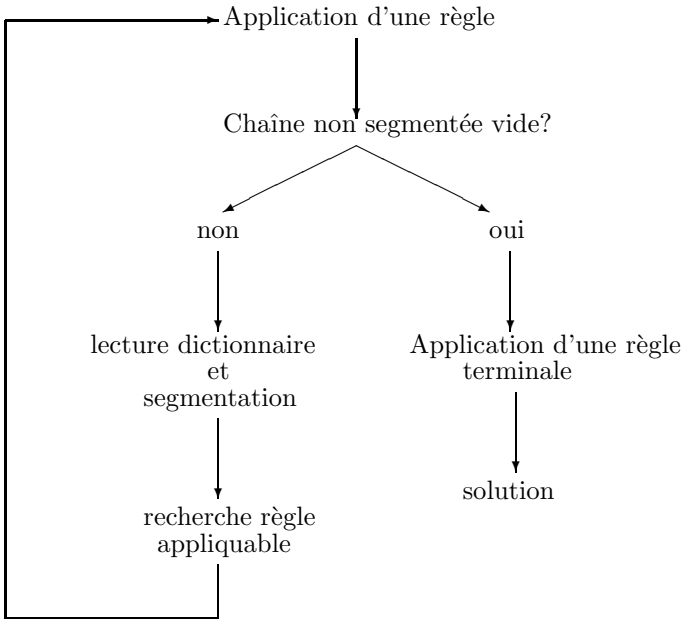
Le but du système OPALE est de définir une transition entre un texte d'entrée et un élément structuré. Cette transition est effectuée par un transducteur d'états finis construit sur la consultation d'un dictionnaire et la segmentation de la chaîne d'entrée. L'élément initial du système OPALE est donc formé d'un couple (élément structure initial, chaîne) et fournit un élément structuré dont la forme dépend de l'analyse de la chaîne. L'élément structuré résultant possède dans toutes ses dimensions la même arborescence et le même étiquetage. Le système OPALE construit une de ces arborescences et cette dernière est dupliquée dans les autres dimensions à la fin de l'application du système. On considère donc ici le calcul de cette arborescence. L'arborescence initiale est constituée d'un point unique que l'on désigne par oméga (EOM). La forme générale de l'application du système OPALE devient alors:

(EOM,chaîne) $\Rightarrow$



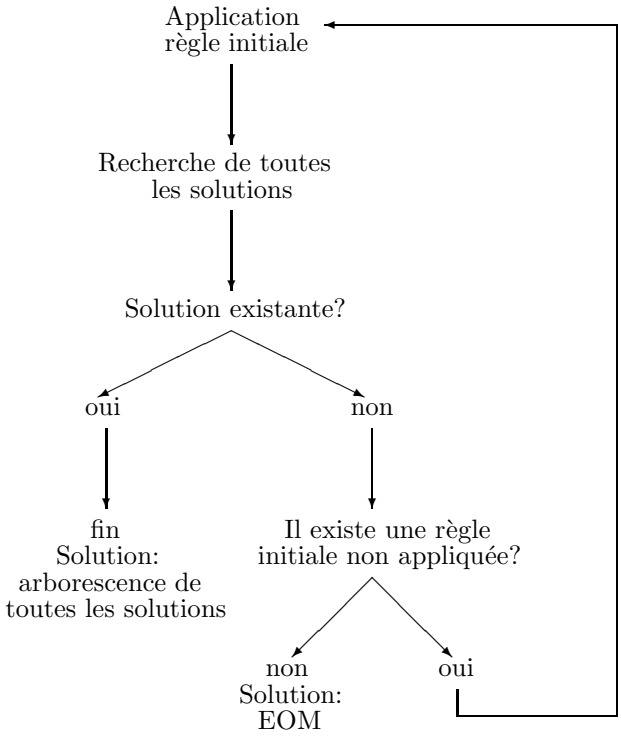
Chaque élément 's_i' correspond à une analyse correcte de la chaîne d'entrée sans obtention de structure complexe. Chaque sous-arborescence de racine EOC correspond à une analyse correcte de la chaîne avec génération de structure complexe.

Le transducteur sous-jacent du système OPALE est un transducteur d'états finis non déterministe et il fournit toutes les solutions possibles. Le calcul d'une solution suit le schéma:



Ce schéma correspond à la recherche d'une solution et donne par conséquent soit la valeur d'une étiquette associée à un point 's_1' dépendant de la racine EOM, soit une structure complexe de racine EOC_i. Il correspond à un fonctionnement correct du transducteur non déterministe. Le résultat global du système OPALE correspond à tous les fonctionnements corrects du transducteur.

Le schéma général du calcul des solutions à partir de la chaîne d'entrée est le suivant:



Les chaînes traitées par le système OPALE lorsque ce dernier est appelé sur un texte sont constituées d'un ensemble de caractères différents des caractères séparateurs (espace, changement de ligne, de page, tabulation, retour chariot). Elles peuvent comprendre des caractères spéciaux ou des ponctuations. Un superviseur produit un appel OPALE pour chaque suite ainsi isolée. L'étiquette associée au point EOM est commune à tous les appels. Certaines fonctions OPALE permettent de modifier cette segmentation

initiale par introduction d'un séparateur virtuel.

Le langage de la grammaire OPALE se décompose en quatre parties:

- Définition de l'environnement.
- Définition des règles initiales.
- Définition de l'adressage des règles.
- Définition des règles.

2.1 Définition de l'environnement

La définition de l'environnement est commune à tous les langages des sous-systèmes. Elle permet d'affecter un nom (1 à 8 caracteres) à la grammaire, de préciser une définition de référence pour les variables utilisées et de définir les variables agrégats.

2.2 Définition des règles initiales

La définition des règles initiales permet de preciser une liste de noms de règles qui seront appliquées systématiquement à chaque analyse (règle correspondant au segment vide). Le résultat du système correspondra donc à l'ensemble des solutions qui débutent par l'application d'une de ces règles initiales. Cette liste peut comprendre deux parties séparées par "/". Dans ce cas, la première partie correspond à la liste à appliquer lors du premier appel du système et la seconde partie la liste à appliquer lors des appels suivants. Si cette liste ne comprend qu'une seule partie cette dernière sera appliquée dans tous les cas.

Règle de syntaxe:

- soit &INIT(R_1,...,R_n/R'$_1$,...,R'$_n$).
- soit &INIT(R_1,...,R_n).

2.3 Définition de l'adressage des règles

La définition de l'adressage des règles appartient au processus interne du transducteur. Le résultat de la lecture du dictionnaire fournit une valeur d'étiquette complexe et une segmentation de la chaîne d'entrée. Cette segmentation sera considérée comme potentiellement valable si au moins une règle de la grammaire est applicable avec ce découpage. Pour déterminer les règles susceptibles d'être appliquées après cette segmentation, il est nécessaire d'avoir un dispositif d'adressage conditionné par la lecture du dictionnaire. Ainsi la valeur de l'étiquette complexe issue du dictionnaire permet de déterminer une liste de règles applicables. L'adressage de cette liste est défini par niveau d'index de la même manière que l'adressage du dictionnaire: une variable quelconque est définie comme un index et, à une de ses valeurs possibles, est associé, soit un niveau d'index supplémentaire, soit une liste de règles potentiellement applicables. Pour définir ces niveaux d'index, deux types de clés sont possibles: le type exclusif réprésenté par des variables de type arithmetique, exclusif, non exclusif, potentiel ou agrégat et le type non exclusif representé par des variables de type non exclusif uniquement. Un niveau d'index de type non exclusif peut fournir un ensemble de listes de règles, alors qu'un niveau d'index de type exclusif ne donne qu'une liste de règles.

Règles de syntaxe:

Les index forment une structure de blocs imbriqués. L'entête du bloc définit la variable utilisée comme index et le bloc lui-même est constitué par une liste des valeurs de variables index associée à l'adressage.

entête de bloc:

&CLEX(ZZZ). ou &CLNEX(ZZZ).

la liste des valeurs de la variable index prend deux formes suivant que cette liste définit des niveaux d'index supplémentaires ou des listes de règles.

Liste définissant des niveaux d'index supplémentaires:
&VAL(ZZZ,WWW).
où WWW est une valeur de la variable ZZZ qui forme l'index.
Liste définissant des listes de règles:
(WWW,....,WWW') $->$ (R_1,...,R_n).
où WWW, ... , WWW' sont des valeurs de la variable ZZZ et
R_1, ... , R_n les noms des règles adressées par ces valeurs.

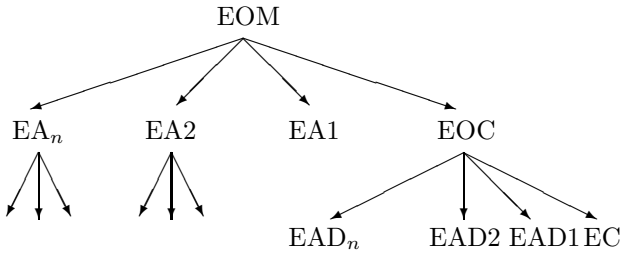
La fin d'un bloc est définie soit par la présence d'une valeur d'index d'un bloc superieur, soit par la fin de la liste d'adressage.

2.4 Définition des règles

La définition des règles est l'élément central de la grammaire OPALE. Elle permet de définir les transitions du transducteur sous-jacent. L'état de ce transducteur est constitué par un ensemble fini d'étiquettes complexes. Chacune de ces étiquettes porte un nom pré-défini. L'évolution de l'état du transducteur correspond à l'évolution d'une ou plusieurs de ces étiquettes. La lecture du dictionnaire de chaînes détermine une étiquette complexe qui constitue le symbole d'entrée du transducteur. Une règle précise donc l'évolution de l'état du transducteur. Cette évolution est subordonnée à deux facteurs: L'adressage d'une règle et la vérification de la condition de la règle adressée. Simultanément à l'évolution de l'état du transducteur, une action sur la chaîne d'entrée et diverses fonctions d'accéleration peuvent être définies à l'aide de fonctions spécifiques. Le transducteur étant non déterministe, il analyse les différentes solutions à l'aide d'une pile (simulation d'un transducteur d'états finis non déterministe par un transducteur à pile). L'état du transducteur est fonction de l'analyse en cours et des solutions précédemment trouvées.

Le point central de cet état est l'étiquette complexe dénommée état courant 'EC' qui définira la solution d'une analyse. Cette étiquette est complètée par les étiquettes complexes suivantes:

1. étiquette attachée à la racine de l'arborescence des solutions: 'EOM'.
2. étiquette attachée à la racine d'une solution composée en cours d'élaboration: 'EOC'.
3. étiquette attachée aux solutions précédentes ou aux racines de ces solutions dans le cas d'une solution complexe: 'EA_i'.
4. étiquette attachée aux différentes composantes d'une solution composée en cours d'élaboration 'EAD_i'.
5. étiquette attachée au segment lu dans le dictionnaire de chaîne 'ED'.



Le nombre maximum d'étiquettes adressables est de dix par niveau (EA1 a EA10 et EAD1 a EAD10). Lorsque, soit la fonction SOL est appliquée, soit une segmentation complète a pu être réalisée, l'étiquette EC est associée à un point et complète l'élément structuré solution.

La définition d'une règle comporte quatre parties:

- une condition d'application.
- une suite d'affectations de variables.
- une suite de fonctions de contrôle.

- une liste de règles complémentaires.

Dans les parties condition et affectation la chaîne d'entrée est repérée par la pseudo variable CHAINE qui peut avoir comme origine la chaîne courante (CC), la chaîne analysée (CA), la chaîne lue dans le dictionnaire (CD) ou la chaîne des séparateurs précédant la chaîne analysée (CS). Les autres paramètres possible de cette pseudo variable ne sont que des constantes (origine : nombre ou caractère entre apostrophes; longueur : nombre, caractère entre apostrophes ou *).

2.4.1 Conditions d'application

La condition d'application est une expression booléene de conditions élémentaires portant sur les valeurs des variables des différentes étiquettes adressables.

2.4.2 Suite d'affectations

La suite d'affectations permet de modifier les valeurs des étiquettes complexes formant l'état courant. Les étiquettes modifiables sont EC, EOM, EOC et ED. La modification de l'étiquette ED est locale et n'est donc valable que pour la suite de l'application de cette règle et des règles complémentaires associées. Cette modification est annulée dès qu'une nouvelle lecture du dictionnaire est effectuée. La valeur affectée à une variable peut être calculée à partir des valeurs de toutes les variables des étiquettes adressables.

2.4.3 Fonctions spécifiques

Les fonctions spécifiques suivent la liste des affectations et permettent d'altérer la chaîne d'entrée ou d'accélérer la recherche d'une solution. Ces fonctions sont au nombre de six:

ART : élimine toutes les segmentations qui contiennent un segment interne au segment lu dans le dictionnaire et concerné par cette règle.

ARD : élimine toutes les segmentations, issues d'une lecture de dictionnaire, qui contiennent un segment interne au segment concerné par cette règle.

ARS : élimine toutes les sous-règles appartenant à la suite des sous-ègles en cours d'exécution.

FINAL : élimine toutes les solutions autres que la première issue de l'application de cette règle.

SOL : définit l'état courant comme fils du point associé à l'étiquette EOC et continue l'analyse en conservant la valeur courante de EC.

SOLV : définit l'état courant comme fils du point associé à l'étiquette EOC et continue l'analyse en conservant la valeur courante de EC. La fonction précédente 'SOL' ne s'appliquera pas dans le cas où la segmentation est complètement réalisée (chaîne courante vide). Par contre cette fonction 'SOLV' s'appliquera dans tous les cas.

PEOC : impose la présence du point "EOC" dans le résultat. Ce point est normalement présent que dans le cas d'une solution complexe comportant plus d'un point. Cette fonction permet de définir une solution simple sous un point "EOC".

FRM : redéfinit le découpage initial du texte. La partie de la chaîne d'entrée déjà analysée formera un mot distinct de la partie restant à analyser. Cette fonction implique un traitement identique à celui effectué avec une fonction 'FINAL'. Le découpage correspondant sera donc considéré comme unique.

NUL : l'analyse ne donnera pas de résultat ni de point EOM correspondant (Sans résultat et sans l'application de cette fonction un point EOM est toujours produit).

TCHAINE : altère la chaîne d'entrée. Cette fonction possède trois paramètres et est basée sur le remplacement d'une sous-chaîne de la chaîne d'entrée par une chaîne donnée. Les deux premiers paramètres définissent la sous-chaîne à remplacer, le premier définissant l'origine de cette sous-chaîne, le second la longueur de cette sous-chaîne. Le troisieme paramètre définit la chaîne remplaçante qui peut être une constante ou une valeur de variable chaîne.

TCHAINE(p_1,p_2,p_3)

- p_1: n, +n, -n, 'x', +'x', -'x'
  origine à gauche nième caractère: n ou +n

  origine à droite nième caractère: -n

  origine dynamique à gauche: 'x' ou +'x', premier caractère x rencontre dans un parcours gauche droite de la chaîne.

  origine dynamique à droite: -'x', premier caractère x rencontré dans un parcours droite gauche de la chaîne.
- p_2: 'chaîne', VAR(parvar), *, *'x', *i dans les deux premiers cas, la chaîne à remplacer doit correspondre à la chaîne définie par la constante ou la valeur de variable; dans les autres cas seule la longueur est prise en compte. Cette longueur est soit un entier (*i), soit calculée par rapport au premier caractère rencontré dans le parcours de la chaîne (*X), soit égal au reste de la chaîne (*).
- p_3: 'chaîne', VAR(parvar)

BALISE : définition de balise. Une balise est une suite de caratères comprise entre deux chaînes spécifiques, la chaîne initiale et la chaîne finale. L'intérieur d'une balise ne doit pas contenir la chaîne finale sauf si elle est précédée d'un caractère d'échapement. Cette fonction a deux paramètres : deux chaînes qui sont définies soit par une constante, soit par une valeur de variable. Lorque cette fonction est appliquée elle reste valable pour tout le reste de l'exécution OPALE et ne peut pas être remise en cause par un autre appel. La présence d'une balise correspond également à un séparateur de mots.

LCTFORM : lecture d'un format. Cette fonction a comme paramètre un nom de format qui doit être affectable par une procédure d'entrée/sortie. Le traitement de cette fonction positionne le format paramètre avec le contenu du fichier d'entrée des formats. Chaque appel lit un format dans l'ordre du fichier. Le nom du fichier de lecture de formats est par défaut MODINT/FORMATS.sygeft, il peut être redéfini dans le fichier paramètre.

ECRFORM : écriture d'un format. Cette fonction a comme paramètre un nom de format qui doit être affectable par une procédure d'entrée/sortie. Le traitement de cette fonction complète le contenu du fichier de sortie des formats avec les valeurs du format paramètre. Le nom du fichier d'écriture de formats est par défaut MODINT/FORMATS.sygsft, il peut être redéfini dans le fichier paramètre.

LCTMESSG : lecture d'une variable chaîne d'un format. Cette fonction a comme paramètre un nom de format qui doit être affectable par une procédure d'entrée/sortie et une variable chaîne de ce format. Le traitement de cette fonction positionne la variable chaîne de ce format paramètre avec le contenu d'un fichier identifié comme fichier d'entrée des messages. Le nom du fichier de lecture de message est par défaut MESSAGE.sygent, il peut être redéfini dans le fichier paramètre.

ECRMESSG : écriture d'une variable chaîne d'un format. Cette fonction a comme paramètre un nom de format qui doit être affectable par une procédure d'entrée/sortie et une variable chaîne de ce format. Le traitement de cette fonction complète le contenu d'un fichier, identifié comme fichier de sortie des messages, avec la variable chaîne de ce format paramètre. Le nom du fichier d'écriture de message est par défaut MESSAGE.sygsrt, il peut être redéfini dans le fichier paramètre.

LCTMESSGL : lecture d'une variable chaîne d'un format. Cette fonction a comme paramètre un nom de format qui doit être affectable par une procédure d'entrée/sortie et une variable chaîne de ce format. Le traitement de cette fonction positionne la variable chaîne de ce format paramètre avec le contenu d'un fichier identifié comme fichier d'entrée des messages. La différence avec la fonction LCTMESSG vient du mode de lecture. La fonction LCTMESSG lit le fichier complètement et tronque éventuellement son contenu. Cette fonction, LCTMESSGL, par contre ne lit qu'une seule ligne à chaque appel, le caractère fin de ligne étant supprimé. Chaque appel de cette fonction lit donc les lignes du fichiers des messages) les unes après les autres. Les deux fonctions traite le même fichier.

2.4.4 Définition des règles complémentaires

La liste des règles complémentaires définit une suite de règles dont le processus d'évaluation correspond à une application exclusive. Cette suite sera testée jusqu'à ce qu'une règle conduise à une solution. Dans ce cas, le reste de la liste est ignorée. Si aucune règle de cette suite ne peut conduire à une solution, la règle appliquée sera annulée et donc sans effet.

Règles de syntaxe:

L'ensemble des règles débute par le mot clé "&REGLES." et se termine par le mot clé "&FIN." qui termine également la définition de la grammaire OPALE. Il est alors possible de définir une liste de règles terminales qui seront appliquées à la fin d'une analyse correcte. Ces règles correspondront donc au segment vide situé en fin de mot. La forme syntaxique est alors:

&FIN(< liste de règles >).

Une règle débute par son nom suivi de ':'. Chaque nom de règle de la grammaire doit être différent des autres. La définition de la règle comporte quatre parties séparées par le symbole '/'. L'identification d'une partie est positionnelle, mais chaque partie peut être absente: c'est le nombre de '/' précédent un élément qui détermine sa nature (condition, affectation, fonctions spécifiques ou liste de règles complémentaires). La première partie définit la condition d'application de la règle. Cette condition suit la syntaxe définie d'une manière générale. La seconde partie définit la liste des affectations. Chaque affectation est separée de la suivante par un ';'. Il ne peut y avoir d'affectation conditionelle et la syntaxe d'une affectation est également conforme à la syntaxe définie précédemment. Les affectations sont effectuées dans l'ordre ou elles sont définies et peuvent influer par conséquent les unes sur les autres. La troisième partie définit la liste des fonctions specifiques. Chaque fonction est separée de la suivante par un ';'. Les fonctions TCHAINE sont également appliquées dans l'ordre où elles sont définies et peuvent également influer les unes sur les autres. La quatrième partie définit la liste des règles complémentaires. Elle est composée d'une liste de noms de règles séparées par des ','. Enfin la définition de la règle se termine par un '.'.

Exemple de grammaire OPALE:

```
&REFER(analys2,grexp).
  varnm=varbm(drv,temp).
  varns=varbs(cat,sousg).
  &INIT(rinit,rinc).
  &CLEX(etat).
    &VAL(etat,1).
      &CLEX(nmfmt).
        ('ba3','ba4') -> (ra3,ra5,ra7).
        ('sm1') -> (rs1,rs22).
        ('bn1') -> (rn1).
    &VAL(etat,2).
      &CLEX(nmfmt).
        ('ba3') -> (rs1,rn1).
        ('sm1') -> (rn1).
    &VAL(etat,3).
      &CLEX(poida).
        (3,5) -> (rinc).
   &REGLES.
    rinit: / dict(EC) = 1 | 2; frm(EC) = CHAINE(CA).
    rinc: /cat(EC)=prp/TCHAINE(0,*,'').
    ra3:seg(EC)=seg->0/varns(EC)=varns(ED);dict(EC)=3;nbr(EC)=sin.
    ra5: (sdna(EC) & sdna(ED) != sdna->0) & (drv(EC) != drv->0) &
              (((pdrp(ED) @< am) | (pdrp(ED) @< an)) |
              ((pdrp(ED) @> nam) & (CHAINE(CC,0,1) != '')) /
               varnm(EC)=varnm(ED) / ART.
    ra7:.
    rs1: seg(EC) = seg->0 // TCHAINE(-0,'EM','ENTE')/ra3,ra7.
    rs22: (seg(EC)=d)&(sdna(EA1)=2)/ varns(EOM)=varns(ED).
    rn1: (seg(EAD3)@>=seg(EA2))&(poida(EOM)>4)///ra3,ra5,rinc.
&FIN.
```

Chapitre 3

Le langage TELESI

Le but du système TELESI est de définir une transition entre des éléments structurés. Cette transition est effectuée par un transducteur à pile composé simulant une grammaire transformationnelle. La forme générale de l'application du système TELESI est la suivante:

Le système TELESI se compose d'un réseau conditionnel de grammaires élémentaires. L'application sur un élément structuré est donc définie par un cheminement dans ce réseau avec en chaque point une application d'une grammaire élémentaire. Ce cheminement est conditionnel et dépend de l'élément structuré initial. Certaines grammaires du réseau sont définies comme initiales et le cheminement débutera par l'une d'entre elles. Les sorties du réseau sont repérées par des marquants et le système définira une transition comme terminée seulement aprés avoir atteint un de ces marquants. Il existe deux types de marquants, le marquant positif %STOP indiquant qu'une transition est correcte, et le marquant négatif %NUL indiquant que la transition est fausse. Dans ce dernier cas, la transition devient nulle et l'élément structuré de sortie est l'élément d'entrée. Ce deuxième type de marquant a un intérêt surtout lorsque l'on utilise les propriétés récursives du réseau et que l'on définit un processus d'analyse. Chaque grammaire est définie par une liste ordonnée de règles transformationnelles et un mode d'application. Ce mode d'application permet une utilisation récursive du réseau, la modification des étiquettes associées aux racines ainsi qu'un parcours de l'élément structuré d'entrée (modification dynamique de l'élément structuré par cheminement dans ce dernier par opposition à la modification statique de l'élément structuré considéré alors comme objet. Toute transformation de l'élément structuré d'entrée est realisée par une règle de transformation. Une grammaire élémentaire est donc un algorithme de Markov étendu aux éléments structurés et le réseau une composition simple ou récursive de ces algorithmes. Chaque règle d'une grammaire définit une transformation par remplacement d'un sous-élément structuré. Cette transformation s'éffectue par la transformation simultanée des arborescences composant l'élément structuré. La définition d'une transformation s'éffectue donc relativement à la définition d'un schéma d'élément structuré qui identifie une partie d'un élément structuré.

3.1 Schéma de reconnaissance

Un schéma de reconnaissance est une identification d'un sous-élément structuré. Si cette identification peut être réalisée dans un élément structuré donné, on dit que le schéma est présent dans cet élément.

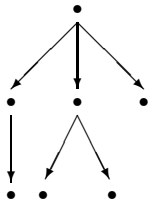
Un schéma d'élément structuré est composé d'un à seize schémas d'arborescences qui devront tous être réalisés pour que le schéma d'élément structuré soit présent. Chacun de ces schémas d'arborescences est attaché à une dimension de l'élément structuré. Deux cas peuvent se produire: une dimension apparait dans le schéma et n'apparait pas dans l'élément structuré et, à l'inverse, une dimension est présente dans l'élément structuré et n'apparait pas dans le schéma. Seule la deuxième forme est possible dans une application TELESI et dans ce cas, la dimension correspondante est ignorée par le schéma.

3.1.1 Schéma d'arborescence

Sous-arborescence:

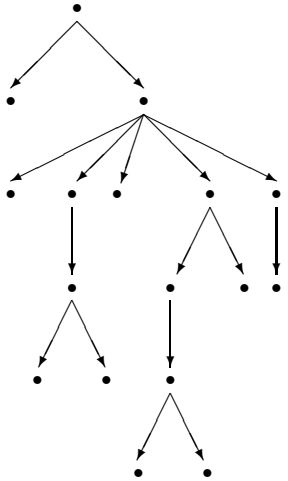
Une sous-arborescence est un ensemble de points d'une arborescence formant eux-mêmes une arborescence ou un ensemble d'arborescences.

L'arborescence:

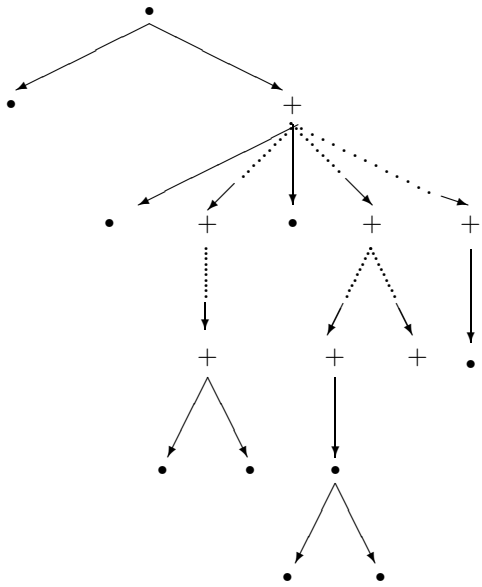


est une sous-arborescence

de l'arborescence:



La décomposition s'éffectue suivant le schéma:



Un schéma d'arborescence est une description d'arborescences. Cette description correspond à une définition d'un ensemble de sous-arborescences potentielles. Si une des arborescences décrites par le schéma est une sous-arborescence d'une arborescence donnée, on dit que le schéma est présent ou s'applique dans cette arborescence. Un schema est défini par une description d'arborescence complètée par un ensemble de contraintes d'applications. Ces contraintes d'applications sont soit d'ordre structurel, soit se rapportent au contenu des multi-étiquettes associées aux différents points. La description du schéma comporte trois parties:

partie structurelle / partie conditionnelle propre / partie conditionnelle inter-sommets

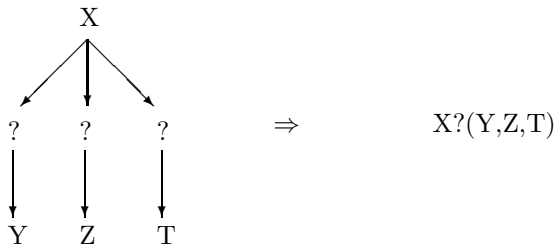
La partie structurelle est décrite dans un langage définissant une arborescence ayant certaines contraintes structurelles. Les différentes contraintes possibles sont:

- définition d'un point : nom composé de lettres ou chiffres : chaque point de la structure d'une dimension doit avoir un nom distinct des autres.
- contrainte de dépendance: symboles: ? ( )
- contrainte de continuité: symbole: *
- contrainte de présence: symbole: %
- contrainte d'ordre: symboles: - ; ,

Contrainte de dépendance

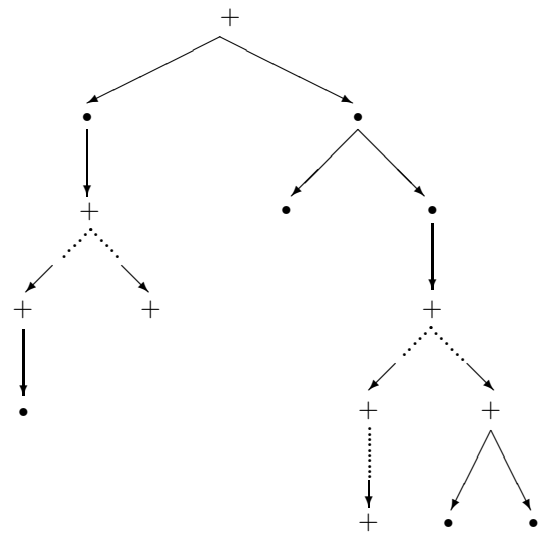
Chaque point du schéma doit être défini par un nom unique pour ce schéma. La définition de la dépendance s'éffectue au moyen de parenthèses: l'ensemble des descendants d'un point suit ce point et est placé entre parenthèses.

Cette définition ne permet de définir que des relations de dépendances strictes. La dépendance généralisée est précisée par un point d'interrogation devant la parenthèse définissant la dépendance:



Exemple:

Le schéma 0?(1(2,3),4(5(6),7)) se trouve dans l'arborescence C:



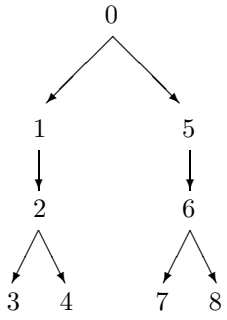
Le schéma peut être un ensemble d'arborescences. Dans ce cas, toutes les composantes de ce schéma sont recherchées au même niveau.

Contrainte de continuité

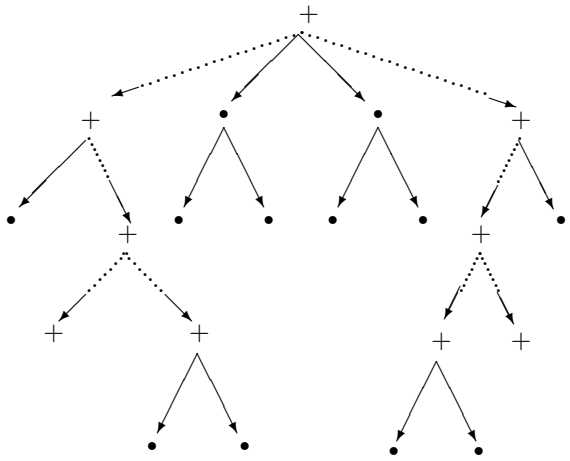
Entre deux points du schéma d'arborescence, il peut y avoir une ou plusieurs arborescences.

Exemple:

Soit le schéma 0(1(2(3,4)),5(6(7,8)))



La reconnaissance de ce schéma dans l'arborescence C suivante n'est possible qu'avec la possibilité d'admettre des arborescences entre les points 1 et 5:



La présence d'une étoile entre deux points précise que le schéma ne doit être reconnu que si ces deux points sont contigus (aucune arborescence entre ces deux points).

Exemple:

Le schéma 0(1(2(3,4)),∗,5(6(7,8))) ne peut être reconnu dans l'arborescence précédente.

Cas particuliers de continuité:

- point feuille (point n'ayant aucun descendant): X(∗)
- point sommet (point n'ayant aucun antécédent): ∗(X)

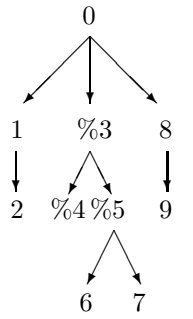
La contrainte de continuité est toujours relative au point qui précéde l'étoile. Cette contrainte de continuité implique l'ordre. C'est à dire que si cette contrainte se trouve entre deux points ces derniers doivent être ordonnés et, si elle se trouve à gauche ou à droite d'une suite de descendants le premier ou le dernier point doit appartenir à un groupe ordonné d'éléments ordonnés.

Contrainte de présence

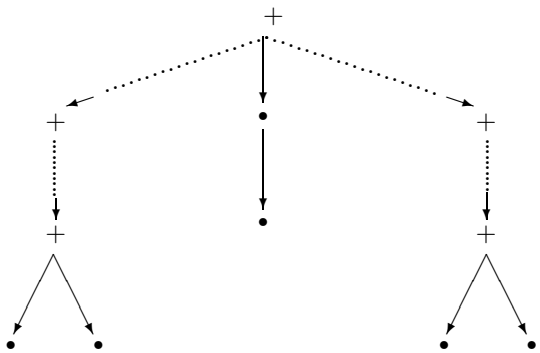
Sans précision supplémentaire, tous les points du schéma doivent être dans l'arborescence pour que le schéma puisse être reconnu. On peut définir un point et ses descendants comme facultatifs et ainsi accepter un schéma même en l'absence de ce point. On précise qu'un point est facultatif en faisant précéder son nom du symbole %.

Exemple:

Soit le schéma A: 0(1(2),%3(%4,%5(6,7)),8(9))

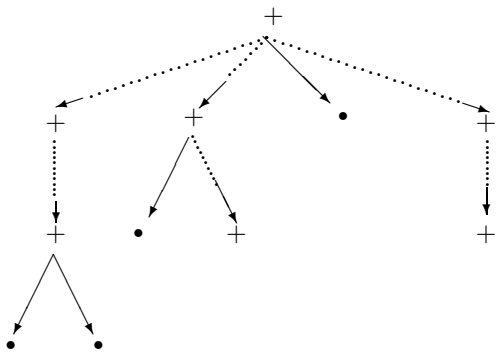


A dans C



A dans C'

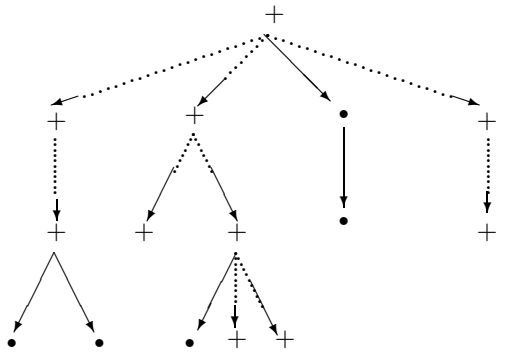
présence de 3 et 4



A dans C"

présence de 3,

4 et 5



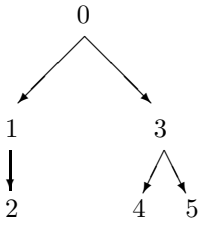
Lorsque le schéma est composé de plusieurs arborescences, le point racine de la première et de la dernière arborescence ne peut être facultatif.

Contrainte d'ordre

Les descendants d'un point peuvent être considerés comme ordonnés ou non ordonnés. Dans le cas ou ces points sont ordonnés, le schéma ne sera reconnu que si l'ordre de désignation des points dans l'arborescence est identique à celui du schéma.

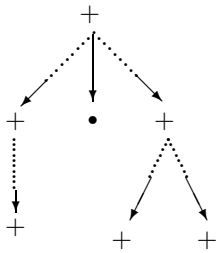
Exemple:

A:



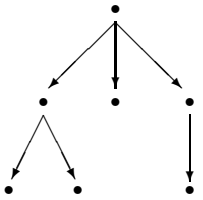
0(1(2),3(4,5))

A dans C



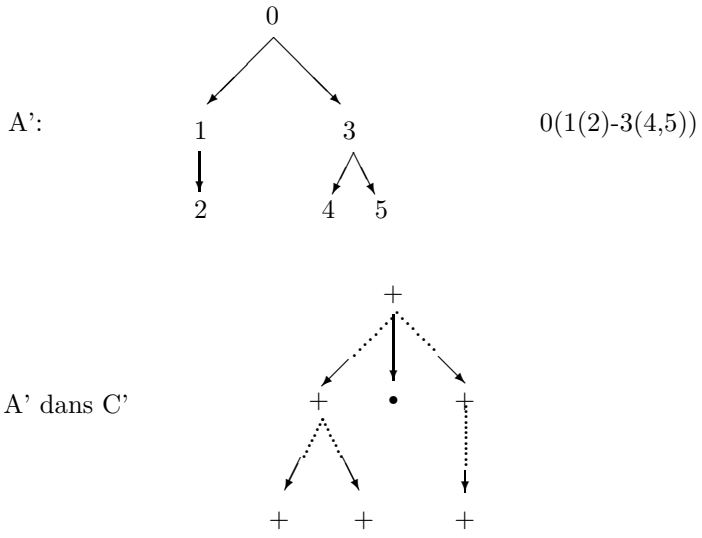
A n'est pas

dans C'



Dans le cas ou les descendants d'un point du schéma de reconnaissance sont non ordonnés, la reconnaissance de ce schéma nécessite seulement la présence de ces points.

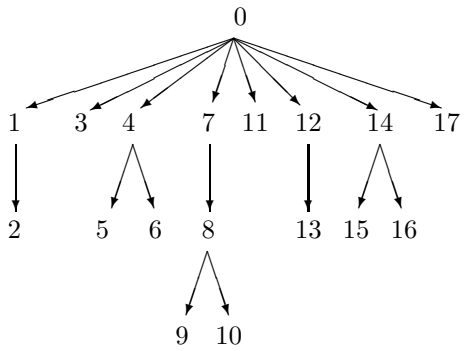
Exemple:



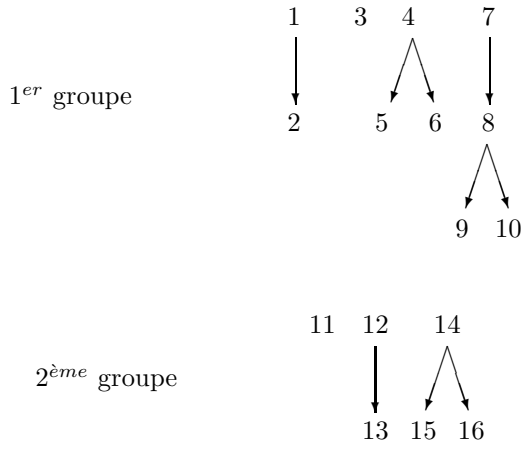
L'ordre ou le désordre des descendants d'un point peut être précisé de trois façons. On considère pour l'ensemble des descendants d'un point un ensemble de groupes ordonnés (chaque groupe sera séparé l'un de l'autre par un point-virgule). Dans un groupe, les éléments peuvent être ordonnés ou non. L'ordre est toujours considéré relativement au point adjacent. Deux points peuvent être non-ordonnés (séparés alors par un tiret) ou ordonnés (séparés alors par une virgule).

Exemple:

soit le schéma 0(1(2)-3,4(5,6)-7(8(9,10));11,12(13)-14(15,16);17)



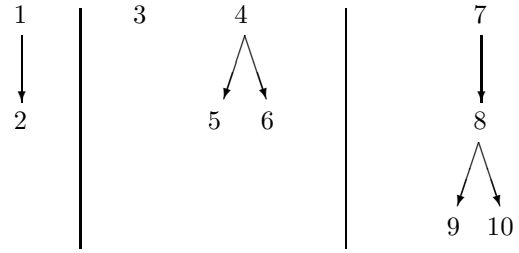
trois groupes:



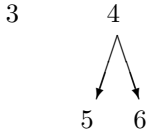
$3^{ème}$ groupe 17

possibilité du 1er groupe:

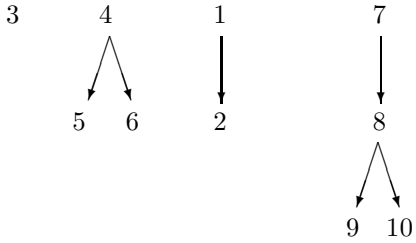
Ordre quelconque entre ces trois blocs:



Le deuxième bloc étant toujours de la forme:

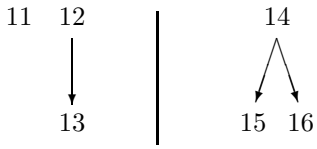


par exemple:

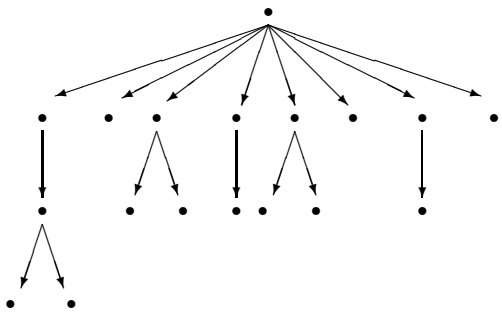


possibilité du deuxième groupe:

deux blocs



L'arborescence suivante contient le schéma ainsi défini:



Lorsque le schéma est composé de plusieurs arborescences, l'ensemble des racines de celles-ci est obligatoirement un groupe d'éléments ordonnés.

Contrainte d'étiquette

Les contraintes d'étiquettes définissent les conditions portant sur les valeurs de variables des multi-étiquettes associées aux points de l'arborescence. On définit trois types de conditions:

- Conditions propres portant sur une étiquette associée à un point.
- Conditions propres portant sur les étiquettes associées aux racines de la liste des arborescences présentes entre deux points du schéma.
- Condition inter-sommets portant sur tout ou partie de l'ensemble des étiquettes associées aux points du schéma et n'appartenant pas au contexte.

Règle d'écriture d'un schéma d'arborescence

écriture linéaire parenthèsée de l'arborescence /
liste des conditions de type 1 ou 2 /
condition inter-sommet

Dans l'écriture de la liste des conditions:

- nom : condition condition sur un point.
- nom$: condition condition sur la liste des racines des arborescences commençant à droite du point nom. Ce point doit être non facultatif et ordonné, ni entouré de points non ordonnés ou facultatifs. Lorsque ce point est dans une dépendance généralisée cette condition porte sur l'ensemble des points et non sur les seules racines. Lorsque ce point appartient à une dépendance généralisée, les points concernés par cette condition sont au même niveau que ce point.
- nom@ : condition condition sur la liste des racines des arborescences commen cant sous le point nom. Pour cette condition il y a les mêmes restrictions que précédemment: Soit il n'existe pas de points descendants du point de référence dans le schéma, soit le premier point défini est non factultatif et ordonné. De la même fa con, si ce point commence une dépendance généralisée cette condition porte sur l'ensemble des points et non sur les seules racines.

Ces conditions peuvent faire intervenir des étiquettes associées à des points extérieurs au schéma par l'intermédiaire des variables références. Deux points du schéma sont nécessairement distincts. La condition pour que ces points fassent référence à la même étiquette est une condition inter-sommets et s'évalue par des variables références (Ref->X = Ref->Y).

Pour les conditions sur les listes (nom$ ou nom@), la définition des conditions est toujours possible. Seul le résultat n'est pas garanti. Par exemple pour le cas ..., X, %Y,*,Z... la contrainte X$ ne sera valable qu'avec la présence de Y. Si l'on souhaite la contrainte avec ou sans présence de Y il faut ajouter la condition Y$ identique à celle de X$. Bien sûr si le schéma est ...,X, %Y, Z la condition entre Y et Z (Y$) doit impliquer la condition X$. Les conditions sont des expressions booléennes de procédures de conditions et de relations de valeurs d'étiquettes. Dans les conditions inter-sommets toutes les variables sont localisées. Dans les conditions propres la localisation est définie par le symbole '*'. Dans ce cas cette localisation peut être absente pour les cas simples (variable simple ou variable repérée).

Exemples:

Condition inter-sommets: (cat(1) = cat(2)) & (gnr(1) & gnr(2) ! = 0)

Condition propre: cat(*) = mas ou simplement: cat = mas

la condition "cat(DICT(*)) = mas" n'est pas simplifiable.

La fonction interne "%EGALSTR" définit une valeur booléene dépendante de la stucture elle-même. Cette fonction ne peut apparaître qu'a l'intérieur d'une condition inter-sommets. Sa définition est la suivante:

%EGALSTR(P_1,P_2) est vraie si les sous-arborescences ayant pour racines les points P_1 et P_2 du schéma sont égales. C'est à dire si ces arborescences ont la même structure et que pour chaque couple de points de cette structure en correspondance les étiquettes associées sont égales. La correspondance de structure est ordonnée:

0(1,2(3)) et 0(2(3),1) sont distinctes.

Choix d'un schéma.

Lorsqu'il existe plusieurs possibilités à la présence d'un schéma dans une arborescence, le schéma choisi est toujours celui qui correspond à la possibilité la plus haute et la plus à droite. Cette contrainte est valable pour tous les points du schéma. Ainsi le point le plus à droite d'un schéma est toujours le plus prioritaire dans la recherche. Lorsque plusieurs contraintes sont définies le choix du schéma est en accord avec la priorité associée à ces contraintes. La recherche d'un schéma ordonné par rapport à la description définie est la contrainte la plus prioritaire. La contrainte de dépendance vient immédiatement après et enfin la contrainte de présence termine la liste des priorités. Ainsi si un schéma existe de façon ordonnée en supprimant un point facultatif ce dernier sera prioritaire par rapport au schéma qui contient ce point mais qui se trouve dans le désordre.

3.1.2 Schéma d'élément structuré

Un schéma d'élément structuré est une suite de schémas d'arborescences, chacun d'eux étant précédé de la dimension concernée par ce schéma. Le schéma sera présent si, pour toutes les dimensions indiquées, le schéma d'arborescence est présent ou l'élément structuré concerné ne contient pas cette dimension.

Règle de syntaxe:

```
[dim]:  partie structurelle /  partie conditionnelle propre /
         partie conditionnelle inter-sommets interne à une dimension /
         partie conditionnelle inter-sommets inter dimensions /
[dim]: ... [dim]: partie structurelle /partie conditionelle propre /
        partie conditionnelle inter-sommets /
        partie conditionnelle inter-sommets inter dimensions
```

Dans le cas où la seule dimension concernée est la première, on peut omettre cette dimension ([1]:). La présence d'un '/' implique la présence d'une condition qui termine le schéma (entre deux caractères '/' il peut ne pas y avoir de condition, le nombre de caractères '/' indiquant le type de condition).

Un nom peut apparaître dans plusieurs dimensions. Dans ce cas, cela signifie que les points correspondants doivent faire référence à la même étiquette.

3.2 Règle de transformation

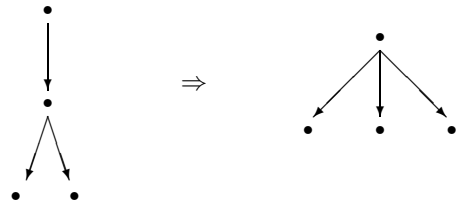
Une règle de transformation se compose d'un schéma de reconnaissance et d'une définition de la transformation de ce schéma. La définition de la transformation s'effectue par une définition de schéma de transformation d'arborescence pour chaque dimension d'un élément structuré.

3.2.1 Transformation d'arborescence"

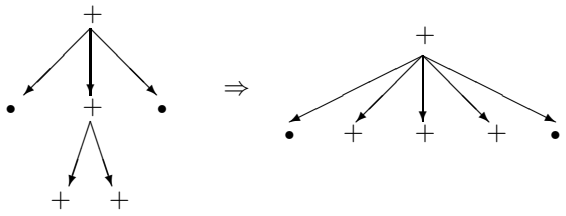
Une transformation d'arborescence est décrite par l'arborescence qui devra remplacer la sous-arborescence reconnue par le schéma. Ce remplacement devra suivre des règles de transfert de listes de sous-arborescences.

Exemple:

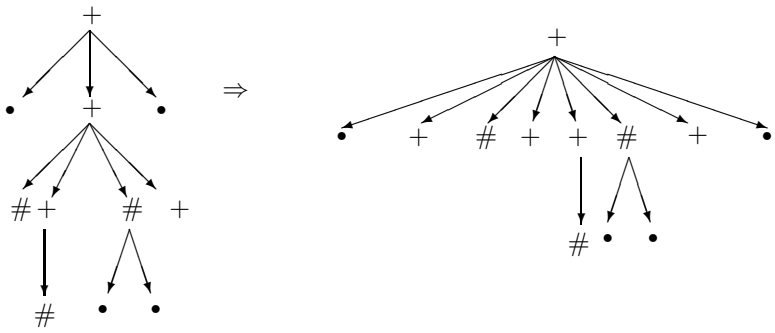
Soit la transformation



Cette transformation modifie l'arborescence suivante par simple remplacement de la sous-arborescence repérée par des symboles "+"



Par contre elle transforme également l'arborescence suivante en transférant une liste d'arborescences dans l'arborescence résultante. Le transfert de ces listes repérées par des symboles "#" doit suivre des règles précises. Ces règles seront définies de deux manières: une déduction automatique et une définition explicite dans le schéma de transformation.



La description d'un schéma de transformation suit les mêmes principes que ceux utilisés pour la définition des schémas de reconnaissance. La description d'un schéma de transformation comporte trois parties:

[dim]: partie structurelle / partie modification propre d'étiquette/
partie modification inter-étiquettes
[dim]: ... [dim]: partie structurelle /partie modification propre /
partie modification inter-étiquettes.

La partie structurelle est décrite dans un langage définissant une arborescence ayant certaines définitions structurelles. Les différentes définitions possibles sont:

- définition d'un point : nom composé de lettres ou chiffres : chaque point de la structure d'une dimension doit avoir un nom distinct des autres.
- définition de dépendance: symboles (et)
- définition de présence: symbole %

  A%B signifie que A sera présent dans la transformation sous réserve de la présence de B dans le schéma de reconnaissance. A%A pourra être ecrit %A

- définition de la recopie d'une structure ayant pour racine un point du schéma d'arborescence de la même dimension: nom du point spécifiant la racine précédé d'un $. Dans ce cas ce nom et les noms de tous ces descendants dans le schéma de reconnaissance deviennent également des noms du schéma de transformation. Leurs étiquettes associées sont donc modifiable. La recopie peut ne concerner que les descendants du point et donc sans le point racine. Dans ce cas le nom de la racine est précédé de $~.

- définition d'une recopie complète de la structure du schéma : symbole $@. Les noms du schéma de reconnaissance deviennent des noms du schéma de tranformation.

- définition de liste: forme $* <,> *$
  Une liste désigne un ensemble de points présent dans la structure à transformée et non identifié dans le schéma de reconnaissance. Elle peut également désigner un ensemble de points produit par des appels spécifiques : analyse morphologiques ou lecture d'un dictionnaire.

  Une liste est définie par trois éléments: le père, le frère gauche et le frère droit de la liste.

  Une liste d'arborescences ayant pour père commun le point X, pour limite (frère) gauche le point Y et pour limite droite le point Z sera ecrite $*X<Y,Z>*$.

  Certains de ces points peuvent être absents (pas de limite gauche ou droite ou pas de père). L'ensemble des descendants d'un point X sera donc défini par $*X<,>*$ que l'on peut écrire $*X*$.

  Dans le cas ou l'un des éléments, gauche ou droit, est absent, cela signifie que la liste commence ou finit à gauche ou à droite des descendants du point X père de la liste.

  Dans le cas où le père est absent, cela correspond à la description d'une forêt d'arborescences et les éléments gauche et droit doivent être présents. Le père est alors symbolisé par '@'.

  Une liste peut être extraite d'une autre dimension que celle de la structure transformée. Dans ce cas la dimension concernée est précisée avant le père:

  $*[ndim]:X<Y,Z>*$

  Au cours de la création d'une liste, la fonction d' étiquetage est inchangée. Il peut alors y avoir prodution d'un étiquetage non injectif. Une liste peut être crée avec création de nouvelles étiquettes : dans ce cas le nom du père de la liste est précédé d'un '!'. Pour chaque point de la liste, la fonction d'étiquetage désigne une nouvelle étiquette dont la valeur est identique á celle de l'étiquette définie par la fonction d'étiquetage de la structure source.
  Trois listes particulieres peuvent être définies:

  la première correspond au résultat de l'appel du sous-système OPALE sur une chaîne de caractères. Cette chaîne de caractères doit être la valeur d'une variable chaîne associée à l'un des points du schéma de reconnaissance. La définition de la liste prend alors la forme: $*&OPALE(noma(nomb))*$

  où:

  - noma: nom de la variable chaîne définissant la chaîne à analyser. Cette variable peut être repérée. Dans ce cas "noma" a la forme: noma1->...nomap.
  - nomb: nom du point du schéma de reconnaissance auquel est associée la variable chaîne.

  La deuxième correspond à la lecture d'un dictionnaire d'étiquette. Un point de cette liste sera défini pour chaque lecture possible du dictionnaire et sera affecté des valeurs du premier masque associé à cette lecture. La définition de cette liste est de la forme: $*&DICT(nom)*$

  où:

– nom: nom du point dont l'étiquette associée constituera l'entrée du dictionnaire.

Dans le cas où la lecture du dictionnaire comporterait une sélection les points de la listes sont ordonnés et en nombre limités. Par défaut ce nombre est de 10 au maximum. Ce nombre peut également être défini comme paramètre, sans pouvoir dépassé 2048. La forme est alors : ∗&DICT(nom,<nombre max>)∗

La troisième correspond à la lecture d'une dimension d'un élément structuré défini dans un fichier. La variable chaine définie en paramètre doit contenir le nom du fichier. Le deuxiè paramètre indique la dimension à lire. La forme est alors : ∗&LECT(noma(nomb),n)∗

où:

– noma : nom de la variable chaîne définissant le nom du fichier. Cette variable peut être repérée. Dans ce cas "noma" a la forme: noma1->...nomap.

– nomb : nom du point du schéma de reconnaissance auquel est associée la variable chaîne.

– n : numéro de la dimension concernée.

La partie modification propre d'étiquettes permet de définir la gestion des étiquettes. Cette gestion comprend:

- la fonction d'étiquetage pour les points du schéma, c'est à dire de préciser pour chaque point du schéma de transformation l'étiquette à associer à ce point. Cette étiquette est construite suivant quatre méthodes principales:
  - la composition de fonction: l'étiquette est déterminée par la valeur de la fonction d'étiquetage sur un point du schéma de reconnaissance. Par extension cette forme comprend la création d'une nouvelle étiquette vide.
  - la création d'une nouvelle étiquette ayant la valeur d'une étiquette associée à un point du schéma de reconnaissance.
  - la recopie d'une valeur d'étiquette issue d'une valeur de format.
  - Le calcul d'une référence d'étiquette: soit une valeur de variable, soit la valeur d'une fonction produisant une référence d'étiquette.
- la désignation d'une étiquette et son évolution. Le nouveau nom utilisé pour cette modification ne peut pas être un nom des schémas de reconnaissance ou de transformation et désigne toujours une étiquette repérée par une variable référence ou la valeur d'une fonction d'étiquette.

La forme syntaxique dépend de la façon dont l'étiquette est construite:

Définition à partir de l'étiquetage d'un point du schéma de transformation:

nom du point du schéma de transformation : vide (création d'une étiquette vide) ou nom du point du schéma de reconnaissance.

Définition à partir de la recopie d'une étiquette associée à un point du schéma de reconnaissance:

nom du point du schéma de transformation : nom du point du schéma de reconnaissance précédé du symbole ! .

Définition à partir de la recopie d'une étiquette définie dans un format:

nom du point du schéma de transformation : nom du format source pour la copie précédé du symbole # .

Définition à partir du calcul d'une référence:

nom du point du schéma de transformation = valeur d'étiquette (valeur de variable référence ou appel de la fonction %ETIQUETTE:).

Définition d'une modification d'étiquette sans traitement de la fonction d'étiquetage :

> @<Nom désignant l'étiquette> : localisation de l'étiquette à modifier (valeur de variable ou de fonction). Dans ce dernier cas une suite d'affectations conditionnelles est obligatoire.

La modification d'étiquette est facultative. Elle est constituée d'une suite d'affectations conditionnelles placées entre parenthèses et tel que chaque variable affectée est sans origine. Les variables-sources dans le calcul des valeurs ont pour origine un point du schéma de reconnaissance précisant implicitement l'étiquette associée à ce point ou le nom d'une étiquette identifiée pour une modification. Au cours de l'évaluation des différentes affectations toutes les étiquettes associées aux points d'un schéma de reconnaissance conservent leurs valeurs. Une localisation particulière (le symbole '$*$') précise l'étiquette en cours d'élaboration. Cette nouvelle étiquette associée à un point ne pourra être référencée par ce point qu'après la fin de l'évaluation des modifications propres d'étiquettes.

Lorsque plusieurs étiquettes sont concernées par des modifications les définitions de modifications sont séparées par des points virgules:

> modification etiquette 1 ; ... ; modification etiquette n

La partie modification inter-étiquettes permet de modifier de nouveau la valeur des étiquettes associées aux points du schéma de transformation mais en tenant compte cette fois du résultat obtenu dans les affectations propres. Cette partie est une suite d'affectations conditionelles où toute variable utilisée (en affectation comme en calcul de valeur) doit être localisée. Cette localisation est définie soit par le nom d'une étiquette nommée, soit par un point du schéma de transformation qui précise implicitement l'étiquette associée à ce point.

Une barre de fraction permet de définir la nature de la zone qui la suit. S'il n'y a pas de modification inter-étiquettes la dernière barre doit être absente. S'il n'y a aucune modification la première barre de fraction peut être omise.

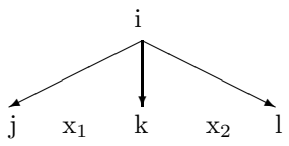
La génération automatique du transfert de listes et de l'association d'étiquettes suit le principe de l'identité de nom. Cette génération automatique peut être supprimée en précisant des noms différents entre le schéma de reconnaissance et de transformation.

L'association d'étiquettes suit strictement l'identité de nom:

> Si deux noms sont identiques dans le schéma de reconnaissance et dans le schéma de transformation ils auront la même étiquette associée.

Le transfert automatique de liste est plus complexe puisqu'il tient nécessairement compte de l'environnement. Une option du compilateur permet de visualiser cette génération automatique de manière à contrôler le fonctionnement d'une règle. Dans le cas où les listes produites par le compilateur ne correspondent pas à celle attendue il est toujours possible, en changeant le nom d'un point du schéma de transformation, de redéfinir le transfert de listes. La définition du transfert produit par le compilateur correspond à un usage courant des transformations. Dans le cas d'un schéma non ordonné ce transfert de liste peut conduire à des copies multiples de tout ou partie des listes. Le principe de l'identité joue ici sur le père de la liste générée automatiquement.

Soit une partie du schéma de transformation: i(j,k,l); les listes générées seront entre j et k, soit x_1, ou entre k et l, soit x_2, ce qui donne éventuellement le schéma i(j,x_1,k,x_2,l).



Si i n'appartient pas au schéma de reconnaissance, alors aucune liste n'est générée.

Si i appartient au schéma de reconnaissance, les générations de x_1 et x_2 sont possibles. Cette génération est conditionnée par le calcul d'une fonction $M_z(p)$. Cette fonction permet de préciser l'environnement d'un point du schéma de transformation par rapport au schéma de reconnaissance. Le calcul de cette fonction est le suivant:

Soit p un point dépendant de i dans le schéma de transformation. Bien sur, i est supposé appartenir au schéma de reconnaissance (dans le cas contraire, la définition de la fonction est sans intérêt, puisqu'il n'y a pas de génération automatique de liste).

- $M_z(p) = p$ si p appartient au schéma de reconnaissance et dépend directement de i.
- $M_z(p) = n$ si p n'appartient pas au schéma de reconnaissance ou ne dépend pas directement de i. n est le point minimum dépendant de p d'une façon générale (donc appartient au schéma de transformation) et dépendant directement de i dans le schéma de reconnaissance.

Nous pouvons alors définir les générations de x_1 et x_2:

- Génération de x_1: $*i<a,b>*$
  - Si $M_z(k)$ est défini:

    s'il existe dans la liste des frères gauches de j (j compris) un élément t tel que $M_z(t)$ soit défini alors il y a génération de x_1 et $b = M_z(k)$ et $a = M_z(t)$. Sinon, il y a génération de x_1 avec a vide.
  - Si $M_z(k)$ n'est pas défini:

    Si k appartient au schéma de reconnaissance et se situe à gauche de i alors il n'y a pas de génération. Si j existe alors si Mz(j) n'est pas défini, il n'y a pas de génération; sinon la borne gauche de la génération éventuelle est Mz(j). Sinon s'il n'existe pas de point j (k est le premier descendant gauche de i) la génération éventuelle commencera à gauche (a vide). Si l n'existe pas alors il y a génération avec abscence de borne droite. Sinon si Mz(l) est défini alors il y a génération avec Mz(l) comme borne droite. Sinon les mêmes conditions sont reportées sur le frère droit de l.
- Génération de x_2: $*i<a,b>*$

  Mêmes possibilités que pour x_1 (en considérant k=l et j=k) sauf dans le cas où l n'existe pas. Dans ce cas, s'il existe dans la liste des frères gauches de k (k compris), un élément t tel que $M_z(t)$ soit défini, il y a génération de x_2 avec $a = M_z(t)$ et $b = 0$. Sinon, il n'y a pas de génération.

La partie transformation peut être vide. Cela correspond à la suppression de tous les éléments définis par le schéma de reconnaissance correspondant.

3.2.2 Transformation d'élément structuré

Une transformation d'élément structuré est définie par un schéma d'élément structuré (partie gauche de la transformation) et par un schéma de transformation d'élément structuré décrivant, pour chaque dimension, le schéma de transformation correspondant. Si un schéma de transformation est décrit dans une dimension, la partie gauche doit nécessairement comprendre pour cette dimension un schéma de reconnaissance. Bien sûr, il est possible que toutes les dimensions précisées dans le schéma de reconnaissance ne donnent pas lieu à un schéma de transformation. Chaque description d'un schéma de transformation doit être précédée de la dimension concernée. Un cas particulier est celui de la première dimension décrite seule. Dans ce cas, la référence à la dimension peut être omise.

La forme syntaxique générale est donc la suivante:

schéma de reconnaissance => schéma de transformation.

Lorsqu'un schéma de transformation est identique au schéma de reconnaissance, il peut être remplacé par des guillemets.

3.2.3 Récurrence de règle

L'application d'une règle peut avoir plusieurs modalités. La plus simple est définie par l'application de la règle; la plus complexe par l'application de la règle complètée par l'application d'un réseau de grammaires.

Toutes ces modalités d'application s'effectuent dans le contexte des grammaires. Une grammaire définit un ensemble de règles. Parmi cet ensemble un sous ensemble de règles est testé et éventuellement appliqué. Ce sous ensemble constitue l'ensemble des règles actives à un instant donné. Cet ensemble de règles est appliqué sur un élément structuré. Les contrôles définis sur une règle portent sur l'ensemble des règles actives et sur l'ensemble des points de l'élément structuré.

La première modalité d'application d'une règle concerne le contrôle des règles actives. Ce contrôle comporte trois cas:

- Seule la règle appliquée reste éventuellement active. (Comme les contrôles associés aux règles appliquées simultanément s'ajoutent, deux contrôles de ce type peuvent conduire à une liste vide de règles actives).
- Seules les règles actives qui appartiennent à une liste déterminée resteront actives.
- Aucune modification de l'ensemble des règles actives. (Ce type de contrôle est utile lorsque l'on souhaite contrôler seulement les points de l'élément structuré).

Chacun de ces contrôles est suivi d'un contrôle sur les points de l'élément structuré. Pour une dimension le point du schéma de transformation qui est mentionné constitue le point repère pour cette dimension. Ce point est suivi d'une liste de points de ce même schéma de transformation. Tous les points de cette liste doivent dépendre du point repère et le point repère peut appartenir à cette liste. Tous les points de cette liste seront bloqués pour les applications futures de cette grammaire. C'est à dire qu'aucun de ces points ne pourra appartenir à un schéma de transformation et qu'aucun schéma de transformation ne pourra dépendre d'un tel point. Ce blocage concernera également les appels récursifs éventuels. Il se terminera avec la fin de la grammaire.

La deuxième modalité correspond à un appel récursif. Un appel récursif a la signification suivante: La règle contenant cet appel sera considérée comme appliquée lorsque les règles ou grammaires appelées en récurrence auront été appliquées. Cet appel récursif complète donc l'application d'une règle et permet de définir ainsi des transformations plus complexes. Il existe trois familles d'appels récursifs:

- Le premier appel récursif concerne un ensemble de règles. Cet ensemble de règles doit appartenir à une grammaire. Lorsque l'ensemble des règles appelées en récurrence constitue la grammaire elle même il est inutile d'énumérer cet ensemble. Dans cet appel les règles ainsi définies seront de nouveau testées et appliquées sur le résultat de la transformation de la règle appelante. Cette application complètera donc la transformation de cette règle.
- Le deuxième appel récursif concerne une grammaire. Cette grammaire sera appliquée sur le résultat de la transformation de la règle appelante.
- Le troisième appel récursif concerne le réseau de grammaires. Cet appel concerne bien sûr une grammaire du réseau mais aussi les grammaires dépendantes de cette première grammaire. Cet appel sera terminé avec l'appel d'une pseudo grammaire %STOP.

Ces trois appels récursifs sont suivis d'un ensemble de contrôles sur les points de l'élément structuré. Ces contrôles sont définis de la même manière que précédemment. Ici le point repère a une signification supplémentaire: il définit la racine de la sous arborescence qui sera traitée par la récurrence. Le compilateur exige qu'il y ait au moins un tel contrôle mais ne vérifie pas la cohérence des appels récursifs. Si plusieurs règles comportent des appels récursifs il est nécessaire que ces appels soient éffectués sur la racine des transformations dans chaque dimension.

Le blocage des points dans le cas d'appel récursif n'est valable que pour cet appel et non pour la grammaire elle même. C'est à dire que ce blocage sera terminé avec la fin de la récurrence.

Les différentes formes sont donc les suivantes:

- $*$: La règle appliquée devient la seule règle active.
- $*$(liste de règles): Les règles actives seront celles qui seront actives au moment de l'application de cette règle et qui appartiendront à cette liste.
- $*(*)$: Aucune modification de la liste des règles actives.
- nom de grammaire : appel récursif de règles. La liste des règles appelées en récurrence est celle définie par cette grammaire.
- nom de grammaire ($*$): même signification
- nom de grammaire (liste de règles) : appel récursif de règles. La liste des règles appelées en récurrence est obtenue par l'intersection de la liste précisée et de la liste définie par la grammaire.
- $nom de grammaire : appel récursif de grammaire. Cette grammaire est considérée dans son ensemble avec son mode d'application I,U,V ou E.
- $nom de grammaire($*$) : même signification.
- $nom de grammaire(liste de règles) : appel récursif de grammaire. Cette grammaire conservera son mode d'application I,U,V ou E et sera limitée dans sa composition à la liste des règles obtenue par l'intersection de la liste définie par cette grammaire et de cette liste.
- @nom de grammaire : appel récursif du réseau de grammaires commençant à la grammaire précisée, le départ ayant la même signification que l'appel "$nom".
- @nom de grammaire ($*$) : même signification.
- @nom de grammaire (liste de règles) : appel récursif du réseau de grammaires commençant à la grammaire précisée, cette dernière étant limitée aux seules règles obtenues par l'intersection des listes des règles définies par la grammaire et de celles énumérées.

Lorsque la grammaire qui définit soit une liste de règles appelées en récurrence soit un appel récursif comporte une structuration, la liste éventuellement concernée est celle obtenue par l'intersection de la liste définie dans la récurrence et la liste applicable de la structure de sous grammaires.

Il est possible de définir plusieurs appels récursifs sur une mème transformation. Dans ce cas il est nécessaire que la suite des points racine de récurrence soit ordonnée de droite à gauche et que chacun de ces points soit indépendant l'un de l'autre. Les différents appels récursifs sont séparés par une barre de fraction. Les récurrences seront appliquées dans l'ordre indiqué.

Exemple:

$$r1(g1(*);[1]:A[2]:A[3]:B/[1]:C[2]:D[3]:A).$$

La regle r1 sera complétée par deux appels récursifs. Dans le premier appel les racines seront les points A, A et B des dimensions 1,2 et 3. Dans le deuxième appel les racines seront C, D et A pour les dimensions 1,2 et 3. Dans la dimension 1 le point A doit être indépendant du point C et se trouver à droite de ce dernier. De même dans la dimension 2 le point A doit être indépendant du point D et se trouver à droite. Dans la dimension 3 le point B doit être indépendant du point B et se trouver à sa droite. Le compilateur ne vérifie pas que ces conditions sont effectivement réalisées. L'appel sera correct que si ces conditions sont vérifiées.

Contrôle sur la grammaire élémentaire ou le réseau:

Deux contrôles supplémentaires peuvent être définis. Le premier fait référence au mode d'application de la grammaire et permet de bloquer une règle pour les applications futures de cette grammaire. Alors que dans le premier contrôle, on doit préciser pour une liste de règles celles qui doivent être conservées, on précise ici celles qui doivent être otées. Bien sûr, cela correspond à un arrêt imperatif de la grammaire lorsque l'on précise toutes les règles. Cette fonction de contrôle est de la forme:

- $HLT : blocage (arrêt) de la grammaire élémentaire.
- $HLT(liste de règles): blocage des règles listées.

Le deuxième contrôle concerne le réseau. Il permet de définir à l'avance le chemin que doit suivre le système dans l'application du réseau. Cette fonction s'effectue en précisant la grammaire élémentaire à appliquer après celle qui est en cours d'exécution. Cette fonction a six variantes:

- $TRF(<nom de grammaire>) : la grammaire prise en compte sera celle issue de la première fonction appliquée.
- $TRF(INV:<nom de grammaire>) : la grammaire prise en compte sera celle issue de la dernière application.
- $TRF(RGL:<nom de grammaire>) : la grammaire prise en compte sera celle associée à l'application de la règle la plus prioritaire.
- $TRF(IRG:<nom de grammaire>) : la grammaire prise en compte sera celle associée à la règle la moins prioritaire.
- $TRF(GRM:<nom de grammaire>) : la grammaire prise en compte sera celle associée à la grammaire la plus prioritaire.
- $TRF(IGR:<nom de grammaire>) : la grammaire prise en compte sera celle associée à la grammaire la moins prioritaire.

La priorité des grammaires est définie par l'ordre de la première apparition de leur nom. Le calcul de la transition s'effectue toujours dans l'ordre d'application des règles. Le résultat final dépend donc de l'historique de l'application des règles d'une même grammaire. Au cours de l'application d'une règle contenant une fonction "TRF" le changement de la définition du transfert s'effectuera ou non en fonction de celui déjà défini et de la variante associée à cette regle.
Dans le cas où les deux fonctions doivent être appliquées, $HLT doit précéder $TRF.

La syntaxe est donc la suivante:

<nom de règle> [<récurrence 1> [/ <récurrence 2>]...)] [<fonction de blocage>] [<fonction de transfert>] {: | .}

3.3 Grammaire élémentaire

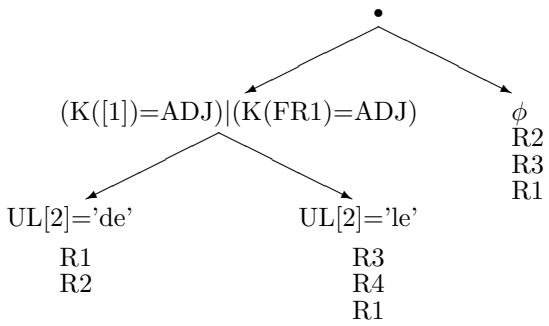
Une grammaire élémentaire est définie par un ensemble de règles et un mode d'application. L'ensemble des règles peut être simple ou hiérarchisé. Dans ce dernier cas, l'ensemble des règles est constitué d'un ensemble de sous-ensembles de règles structuré en arborescence conditonnelle. Chaque point de cette arborescence qui n'est pas une feuille, est étiqueté d'une condition portant sur l'ensemble des racines des arborescences de l'élément structuré. Les points feuilles sont constitués par des ensembles de règles. Le sous-ensemble selectionné sera le premier sous-ensemble gauche qui aura satisfait toutes les conditions associées aux points anterieurs (les conditions associées aux points du chemin allant de la racine à cette feuille sont toutes satisfaites). Deux éléments permettent de définir cette arborescence:

- création d'un sous-niveau: &SGRM: condition.
- fin d'un niveau: &FGRM.

Le repère des étiquettes racines dans la condition s'effectue par le numéro de la dimension: [1] à [16]. La condition peut être vide et dans ce cas elle est considérée comme satisfaite.

Exemple:

La structure:



est définie de la façon suivante ("FR1" est un nom de format):

```
&SGRM: (K([1]) = ADJ)$|$(K(FR1) = ADJ).
  &SGRM: UL([2]) = 'de' .
      R1.
      R2.
  &FGRM.
  &SGRM: UL([2]) = 'le'.
      R3.
      R4.
      R1.
  &FGRM.
&FGRM.
&SGRM:.
  R2.
  R3.
  R1.
&FGRM.
```

La définition de niveau dans une grammaire élémentaire implique que celle-ci ne peut être appelée en récurrence avec une liste de règles. En effet la liste de règles d'une grammaire comportant des sous-grammaires dépend de l'élément structuré manipulé. Il n'est donc pas possible de faire référence à cette liste au moment de la compilation. Les règles des sous-grammaires subissent la même restriction. Tous les appels de récurrence qui ne font pas explicitement référence à une liste de règles sont bien sûr possibles.

Une grammaire élémentaire est toujours introduite par un des deux mots clés:

&GRAM: pour définir une grammaire élémentaire.

&ENTREE: pour définir une grammaire élémentaire qui pourra être la première grammaire du réseau à appliquer. Les paramètres d'applications comporteront nécessairement une référence à une telle grammaire.

Le mode d'application d'une grammaire comporte deux parties: la première se réfère à l'élément structuré et la seconde aux racines des arborescences de l'élément structuré.

La première partie du mode d'application concerne le fonctionnement des règles de cette grammaire et les appels récursifs de grammaires éventuels.

La seconde partie se rapporte à la modification préalable à l'application de la grammaire des étiquettes des racines des arborescences de l'élément structuré.

L'ensemble des règles d'une grammaire est ordonné (ordre défini par l'écriture des règles). L'ensemble de ces règles est appliqué simultanement sur un élément structuré au cours d'une application de la grammaire. Cette application simultanée suit la règle de non-chevauchement:

- Si deux règles sont applicables sur un même point, seule la règle de priorité supérieure est appliquée.
- Si une règle est applicable sur un point, aucune règle de priorité supérieure n'est applicable sur un autre point dépendant de ce point ni sur un point antécédant direct de ce point (points du chemin allant de la racine à ce point).

La grammaire élémentaire est de nouveau appliquée sur le résultat de cette première application. Ce processus itératif se poursuit jusqu'à l'obtention d'une condition d'arrêt. Cette condition d'arrêt se produit notamment lorsqu'aucune règle n'est plus applicable. Le mode d'application de la grammaire élémentaire détermine également cette condition d'arrêt.

Les différents modes d'application d'une grammaire élémentaire sont les suivants:

Iteratif : seul mode indécidable (peut conduire à une application infinie de la grammaire élémentaire). Les règles de la grammaire sont testées et appliquées indéfiniment. Le processus s'arrête lorsqu' aucune règle n'est plus applicable. On peut quand même contrôler ce mode par les règles elles-mêmes (fonction $HLT et contrôle de l'itération qui porte sur les règles et sur les points de l'élément structuré).

notation: I

Unitaire : La liste des règles est testée et appliquée au maximum un nombre n fini de fois ($n < 8192$). L'arrêt de cette grammaire peut donc résulter soit d'une non applicabilité des règles, soit de la fin des n applications.

notation: U(n) ou U pour U(1)

Variable : Même signification que pour le mode d'application unitaire mais avec un nombre n fonction du nombre de points de l'élément structuré d'entrée. Si on considere m le nombre de points de toutes les arborescences de l'élément structuré, alors n est donné par la formule:

$n = \min(8191 , k * (m/80+1))$.

notation: V(k) ou V pour V(1)

Exhaustif : La grammaire est appliquée de façon itérative, mais à chaque pas de cette itération, les règles appliquées sont éliminées pour les applications futures. Le nombre maximum d'itérations est donc le nombre de règles de cette grammaire (cas où, à chaque itération, une et une seule règle de la grammaire est applicable).

notation: E

Le mode d'application d'une grammaire élémentaire comporte également un appel récursif de grammaire. Ce mode récursif comporte deux composantes définies par les récurrences sur arborescence: récurrence verticale et récurrence horizontale. En considérant une arborescence de l'élément structuré (les récurrences ne s'effectuent que sur une dimension de l'élément structuré), la grammaire courante s'applique sur une sous-arborescence complète de cette arborescence (composée de tous les points de cette arborescence dépendant de la racine de l'application).

L'appel vertical est effectué avant l'application de cette grammaire et permet de modifier, par l'intermédiaire de la grammaire appelée en récurrence, la sous-arborescence complète ayant comme racine le fils gauche de la racine de l'arborescence traitée par la grammaire courante.

L'appel horizontal est effectué après l'application de cette grammaire et permet de modifier la sous-arborescence complète ayant comme racine le frère droit de la racine de l'arborescence traitée par la grammaire courante.

Bien sûr, ces appels peuvent faire référrence à la même grammaire et définir ainsi un cheminement dans

l'élément structuré manipulé.

La dimension dont l'arborescence sert de support à la recurence peut être omise. Dans ce cas la première dimension est utilisée comme support de récurrence.

notation: <Nom_de_grammaire>(<mode d'application>, [<dim>] <appel vertical>,<appel horizontal>).

où <dim> a la forme: [n]:

De la même façon que l'appel récursif de règle l'appel récursif de grammaire comporte deux modes d'appel : l'appel simple et l'appel généralisé. Avec l'appel simple la récurrence se termine avec la fin de la grammaire appelée en récurrence. Par contre, avec l'appel généralisé, l'appel se termine avec un arret du réseau (pseudo-grammaire "%STOP").
On indique un appel généralisé en faisant précédé le nom de grammaire du caractère "@".

La seconde partie du mode d'application concerne le traitement des étiquettes associées aux différentes racines des arborescences de l'élément structuré. Elle est constituée d'une affectation conditionnelle de ces étiquettes. Cette affectation a lieu avant l'application des règles de la grammaire élémentaire (et donc après l'appel récursif vertical éventuel).

notation: Nom_de_grammaire(mode):affectation des racines.

Pour affecter une racine d'une dimension cette affectation doit être précédée de la dimension concernée. Dans le cas où cette dimension n'est pas précisée il s'agit implicitement de la première dimension.

Exemple:

&GRAM: exp(E): [1]: k=k([2]); UL=UL([3]) [2]: k=0 [3]: UL=UL([1]). &GRAM: nexp: k=k([2]); UL=0.

3.4 Réseau TELESI

Une grammaire TELESI est constituée d'un réseau de grammaires élémentaires. Chaque point de ce réseau correspond à une grammaire élémentaire qui sera appliquée au cours du cheminement dans le réseau défini par l'élément structuré d'entrée. Une grammaire élémentaire de ce réseau peut avoir plusieurs significations vis-à-vis du cheminement:

- point d'entrée: un cheminement du réseau commencera toujours par une grammaire élémentaire d'entrée.
- point courant.
- pseudo grammaire d'arrêt:
  - arrêt correct du cheminement: %STOP.
  - arrêt incorrect du cheminement (et donc retour arrière) : %NUL.

Après chaque grammaire élémentaire, on doit indiquer l'ensemble des grammaires suivantes du réseau. L'ensemble d'arcs de ce réseau est ordonné et définit ainsi le cheminement suivi. Un arc peut être conditionnel ou non. Dans le cas où l'arc est conditionnel, la grammaire suivante correspondante sera appliquée si la condition est vérifiee. Cette condition porte sur la présence ou l'abscence d'un schéma dans l'élément structuré manipulé. Le premier arc dont la condition est positive définira l'application. Bien sûr, dans le cas d'un retour arrière, cet arc sera considéré comme défectueux et le premier arc suivant sera appliqué. Chaque élément de la liste des arcs de ce réseau a l'une des formes suivantes:

- arc conditionnel avec présence de schéma: $--> $ Nom de grammaire : schéma .
- arc conditionnel avec abscence de schéma: $--> $ Nom de grammaire !: schéma.
- arc inconditionnel: $--> $ Nom de grammaire .

Un élément structuré sera transformé par une grammaire TELESI après l'application de toutes les grammaires appartenant à un cheminement correct issu d'une grammaire d'entrée et se terminant à la pseudo-grammaire %STOP. Ce chemin est fonction, entre autres, de la grammaire d'entrée qui, elle, est un paramètre d'application du système TELESI. La fin d'une récurrence générale de règles suit les mêmes règles que l'application globale mais débute par une grammaire quelconque, celle précisée dans la récurrence. Cette récurrence générale, pour se terminer correctement, doit donc aboutir sur la pseudo-grammaire %STOP.

Trois grammaires spéciales permettent de définir des traitements spécifiques de l'élément structuré:

La première définit un échange des arborescences de deux dimensions. Cette pseudo-grammaire est définie par:

%ECHG[I,J]

Où I et J sont les dimensions concernées par l'échange. Un arc associé à cette pseudo-grammaire est toujours considéré comme non satisfait, et donc les arcs suivants seront testés une fois l'échange réalisé. L'échange peut être conditionné par la présence ou l'absence d'un schéma.

Exemple:

Les deux arcs inconditionnels suivants permettent d'effectuer un échange des dimensions 2 et 3 et d'appliquer la grammaire exp:

−− >%ECHG[2,3].
−− >exp.

La deuxième grammaire permet de recopier l'arborescence de la dimension I dans la dimension J. Cette deuxième pseudo-grammaire se définit par:

%ECR[I,J]

La troisième grammaire permet également une recopie de l'arborescence de la dimension I dans la dimension J. La différence porte sur la fonction d'étiquetage. Dans le cas de l'application de la grammaire précédente les étiquettes associées à deux points en correspondance sont identiques (même étiquette). Dans le cas de l'application de cette troisième grammaire les étiquettes sont simplement égales (différentes et ayant la même valeur). Cette troisième grammaire est définie par:

%ECRIMG[I,J]

Les conditions associées aux arcs portent sur le résultat de la grammaire élémentaire et sont testées avant toute application de grammaire ou pseudo-grammaire. Par conséquent l'effet d'une grammaire ou d'une pseudo-grammaire est sans influence sur les conditions associées aux arcs.

Une condition dynamique peut être associée à chaque grammaire ou pseudo-grammaire. Cette condition dynamique correspond à l'application d'une grammaire sur l'élément structuré courant. Si cette application conduit à un résultat positif alors la grammaire ou pseudo-grammaire correspondante sera appliquée. Dans le cas contraire cette grammaire ou pseudo-grammaire sera ignorée. Cette condition s'ajoute aux arcs conditionnels ou non.

Cette grammaire de test ne modifie pas l'élément structuré courant. C'est à dire que l'application de la grammaire ou pseudo-grammaire s'effectue sur l'élément structuré courant avant toute modification éventuelle de la grammaire de test. Pour cette ou ces dernières les modifications effectuées sont temporaires.

On indique cette grammaire de test après la grammaire ou pseudo-grammaire dépendante. Le caractère de séparation est la barre de fraction:

arc inconditionnel:

−− > G1/G2.

La grammaire G1 sera appliquée sur l'élément structuré courant si G2 appliquée sur ce même élément structuré conduit à un résultat (sortie par la pseudo-grammaire %STOP).

arc conditionnel:

−− > G1/G2 : schéma.
−− > G1/G2 !: schéma.

La présence du schéma est préalablement testée. Si le schéma est présent (premier cas) ou absent (2ème cas) le test sur le resultat de la grammaire G2 sera effectué. Si ce test est positif la grammaire G1 sera appliquée. Dans le cas contraire la grammaire G1 sera ignorée.

Les grammaires sont ordonnées de façon à définir une priorité utilisée par la fonction "TRF" associée à une règle. Cet ordre est simplement défini par l'ordre d'apparition des noms des différentes grammaires. On peut définir un ou plusieurs noms de grammaires avant leurs descriptions effectives par la déclaration:

&NOMGRAM: <liste noms de grammaires> .

Dans la liste des noms de grammaires chaque élément est séparé du suivant par une virgule. Cette définition de liste ne peut apparaître que devant la définition d'une grammaire (avant les mots clés &GRAM ou &ENTREE).

3.5 Procédures

Dans une grammaire TELESI, il peut apparaître deux types de procédures: Les procédures conditionnelles et les procédures d'affectations. Ces procédures se rapportent uniquement au traitement des étiquettes associées aux points des éléments structurés. Elles doivent être définies en tête du fichier contenant la définition de la grammaire et ne peuvent être imbriquées (une procédure ne peut comporter dans sa définition un appel à une autre procédure).

Syntaxe:

<nom de procédure> : <corps de procédure> .

ou

<nom de procédure> (<liste des paramètres>) : <corps de procédure> .

3.5.1 Procédures conditionnelles

Une procédure conditionnelle peut être conditionnelle propre ou conditionnelle inter-sommets suivant qu'elle comporte ou non des paramètres. Dans le cas ou cette procédure est conditionnelle propre, le symbole "*" définit l'origine de l'étiquette concernée.

Une procédure conditionnelle propre peut apparaître dans les conditions propres des schémas d'arborescences et une procédure inter-sommets dans les conditions inter-sommets.

La définition d'un ensemble de procédures conditionnelles débute par l'identificateur:

&PROC: CONDITION. ou &PROC: CDT.

3.5.2 Procédures d'affectations

Une procédure d'affectation est constituée d'une liste d'affectations conditionnelles. Elle peut apparaître dans toutes les modifications d'étiquettes et donc être paramétrée ou non. Les affectations de cette procédure sont toutes non localisées et ainsi une procédure d'affectation ne peut apparaître lors des affectations inter-sommets. Deux localisations particulières peuvent être définies dans les procédures d'affectations:

- La première est définie par le symbole "$*$" et représente l'étiquette en cours d' élaboration.
- La seconde est définie par le symbole "#" et représente l'étiquette initiale modifiée par la procédure. Cette dernière localisation n'est accessible qu'aux procédures sans paramètre.

La définition d'un ensemble de procédures d'affectations débute par l'identificateur:

&PROC: AFFECTATION. ou &PROC:AFCT.

3.5.3 Procédures d'entrée/sortie

Il existe deux types de procédures spécifiques qui permettent de définir des entrées/sorties dans une grammaire TELESI. Ces entrées/sorties s'effectuent toujours par la lecture ou l'écriture d'un format de type spécifique. Les procédures déterminent le format qui sera le support de la procédure. Pour la procédure d'entrée le format sera positionné à la valeur du contenu d'un fichier. De même la procédure de sortie produira l'image binaire de la valeur du format dans un fichier. Les noms de ces fichiers peuvent être défini dans le fichier paramètre. Les noms "MODINT/FORMATS.sygeft" et "MODINT/FORMATS.sygsft" seront utilisés si le ou les nom ne sont pas définis dans le fichier paramètre. La lecture ou l'écriture de ces fichiers sera réalisée par l'appel des procédures d'entrée/sortie ainsi définies et peut donc être réalisée au cours d'une condition ou d'une affectation. La définition des procédures d'entrée/sortie doit se faire avant toutes les autres. Bien sûr il est possible de ne pas définir de procédure d'entrée/sortie ou de n'en définir qu'une seule. La syntaxe est la suivante:

&PROC: ENTREE: <nom procédure> (<nom du format d'entée >).

&PROC: SORTIE: <nom procédure> (<nom du format de sortie >).

Exemple d'utilisation:

&PROC: SORTIE: for_sort(fomat_s).

L'appel "for_sort" ou "for_sort()" permettra d'écrire l'image binaire du format "format_s" dans le fichier "MODINT/FORMATS.sft". Bien sûr les variables de ce format pourront être préalablement modifiées.

Lorsqu'une variable chaîne ou externe complète le nom du format (entrée ou sortie) les traitement se font sur des chaînes. Le fichier concerné est alors le fichier des messages et seule la valeur de la variable chaîne du format défini est concerné par cette opération. Exemple d'utilisation:

&PROC: SORTIE: for_mes(fomat_s,message).

L'appel "for_mes" ou "for_mes()" permettra d'écrire la valeur de la variable message du format "format_s" dans un fichier. Bien sûr cette variable de ce format pourra être préalablement modifiée. Lorsqu'il s'agit d'une variable externe la fonction concerne la forme binaire (suite d'octets associés à cette variables). Les noms "MESSAGE.sygsrt" ou "MESSAGE.sygent" seront utilisés si le ou les nom ne sont pas définis dans le fichier paramètre. Les différentes écritures des messages sont concaténées dans le fichier de sortie des messages. La lecture peut être globale (cas par défaut, un appel positionne la variable chaîne au contenu du fichier, limitée à la longueur maximale d'une variable chaîne) ou ligne par ligne. Dans ce dernier cas chaque appel positionne la variable chaîne au contenu de la ligne courante qui progresse à chaque appel et la fin de ligne est supprimée.

Il ne peut y avoir qu'au plus deux procédures d'entrées et deux procédures de sorties. Lorsqu'il y a plusieurs procédures elles doivent être séparées par un point virgule. Il est possible de répéter le type (SORTIE: ou ENTREE:) avant chaque définition de procédure. Il ne peut y avoir au plus qu'une procédure d'entrée/sortie de chaque type (Il y a donc quatre procédure au maximum). Pour la lecture ligne à ligne du fichier des messages le type est "ENTREEL:".

3.6 Grammaire TELESI

Une grammaire TELESI comporte trois parties: la définition de l'environnement, la définition des procédures et la définition du réseau de grammaires.

Chaque grammaire TELESI est supposée s'appliquer sur un élément structuré résultant, soit de l'application du sous-système OPALE, soit de l'application d'une autre grammaire TELESI. La définition de l'environnement détermine uniquement l'espace de référence des étiquettes associées aux points de l'élément structuré. Le nombre de dimensions de cet élément est un paramètre d'appel de la grammaire.

Trois cas peuvent se produire au cours de cet appel:

- Le nombre de dimensions de l'élément structuré résultant est le même que celui défini par l'appel. L'élément structuré est alors transmis simplement au sous-système TELESI.
- Le nombre de dimensions de l'élément structuré résultant est supérieur à celui défini par l'appel. L'élément structuré transmis est alors limité aux dimensions nécessaires. Il est important de noter que les dimensions transmises sont fixes et correspondent toujours aux premières dimensions.
- Le nombre de dimensions de l'élément structuré résultant est inférieur à celui défini par l'appel. Dans ce dernier cas, pour définir l'élément structuré transmis, une fonction de duplication circulaire est executée. Si n est la dimension de l'élément structuré résultant et m la dimension d'appel, l'arborescence de la première dimension sera recopiée dans la dimension n+1, la deuxième dimension dans la dimension n+2, et ainsi de suite jusqu'à m.

La définition d'une grammaire TELESI se termine toujours par le symbole "&FIN.".

Exemple de grammaire TELESI:

```
&REFER(test,gramtl).
  &GRAMMAIRE.
    &ENTREE: constr(U).
        rmisen: [1]: 0(1(*)) [2]: a(b(*)) => [1]: 1 [2]: b.
  --> conshp.
  &GRAM: conshp(I).
    r0$HLT(r0): [1]: 0( * ,1, * ,2, * ,%3) [2]: 0(*,1,*,2,*,%3)  =>
            [1]: 0(1(2,%3)) [2]: 0(1,A,2,%3).
    r1: [1]: 0(1?(2( * )), * ,3, * ,%4) [2]: 0(A, * ,2) / A: chaine=''  =>
            [1]: 0(1(2(3,%4))) [2]: 0(2,A).
    r3$HLT: [2]: 0(A) / A: chaine=''  =>  [2]: S( *0* ,B) / S:0; B:A.
  --> tas.
  &GRAM: tas(U(1),[1]:tas,tas).
    rtasu: [1]: $*$(0(*,1(*),*)) // chaine(0) < chaine(1) [2]: x(0,1) =>
            [1]: A(B) / A:1; B:0
            [2]: y(*x<,0>*,A,*x<0,1>*,B,*x<1,>*) / y:x; A:1; B:0.
    rtasm1(tas;[1]: B [2]:*):
            [1]: *(0(1,2)) // (chaine(0) < chaine(1)) & (chaine(1) > chaine(2))
            [2]: x(0,1) =>
            [1]: A(B(*1*),C(*2*)) / A:1; B:0; C:2
            [2]: y(*x<,0>*,A,*x<0,1>*,B,*x<1,>*) / y:x; A:1; B:0.
    rtasm2(tas;[1]:C [2]:y):
            [1]: *(0(1,2)) // (chaine(0)< chaine(2)) & (chaine(2) > chaine(1))
            [2]: x(0,2) =>
            [1]: A(B(*1*),C(*2*)) / A:2; B:1; C:0
            [2]: y(*x<,0>*,A*x<0,2>*,C,*x<2,>*) / y:x; A:2; C:0.
  -->tri.
  &GRAM: tri(I).
    RD(tas;0):
            [1]: *(S(0?(1(*))))
            [2]: *(x(*,0,1,*,B))/ B: chaine=''
            =>  [1]: S(0) / 0:1
             [2]: y(1,*x<0,1>*,B,0,*x<B,>*)/ y:x.
    RFT:
            [1]: *(S(1(*)))
            [2]: *(x(*,1,*,B)) / B: chaine=''
            =>  [1]: S
            [2]: x(1).
   --> %STOP.
  &ENTREE: constr2(U).
     rmisen.
  --> conshp2.
  &GRAM: conshp2(I).
    r0$HLT(r0).
    r1.
    r3$HLT.
  -->tass.
  &GRAM: tass(U(1),[1]:tass,tass).
    rtassm(tass;[1]:B [2]:y):
            [1]: *(0(1-%2)) // (chaine(0) < chaine(1)) & (chaine(1) > chaine(2))
            [2]: x(0,1) =>
            [1]: A(*0*,B(*1*),%2) / A:1; B:0
            [2]: y(*x<,0>*,A,*x<0,1>*,B,*x<1,>*) / y:x; A:1; B:0.
  -->tri2.
  &GRAM: tri2(I).
    RD2(tass;0):
            [1]: *(S(0?(1(*))))
```

```
            [2]: *(x(*,0,1,*,B))/ B: chaine=''
            =>    [1]: S(0) / 0:1
            [2]: y(1,*x<0,1>*,B,0,*x<B,>*)/ y:x.
    RF2:  [2]: 0(1) / 1: chaine = '' => [2]:0.
  -->%STOP.
&FIN.
```

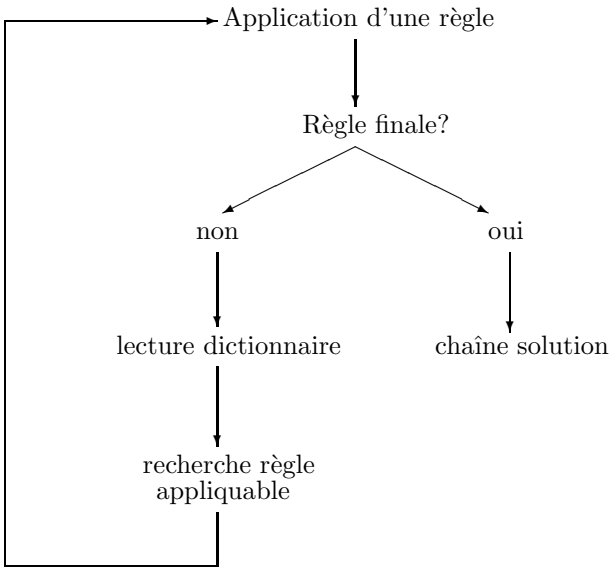
Chapitre 4

Le langage AGATE

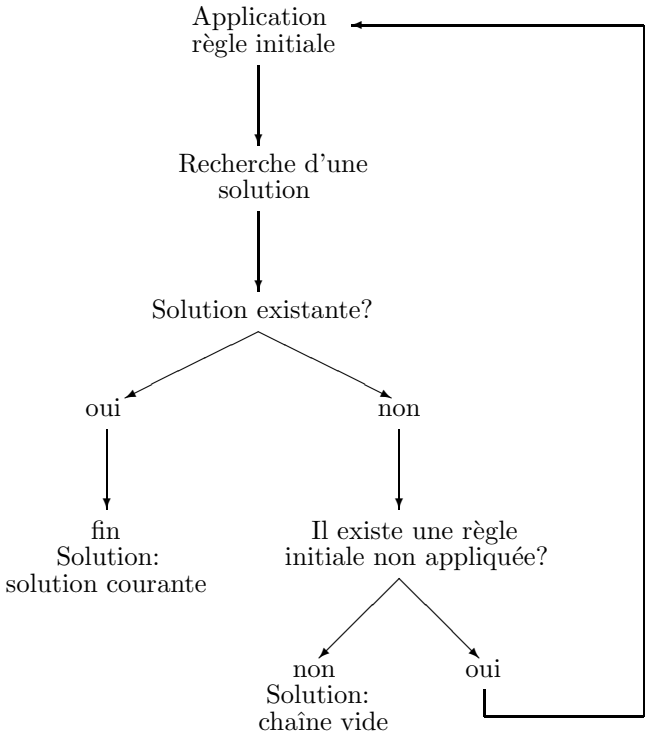
Le but du langage AGATE est de définir une transition entre une multi-étiquette associée à un point d'un élément structuré et une chaîne de caractères.

EOM → chaîne

Le transducteur sous-jacent du système AGATE est comme pour le système OPALE un transducteur d'états finis non déterministe. Il fournit la première solution possible (dans le système OPALE toutes les solutions sont prises en compte). Le calcul de cette solution suit le même schéma que celui défini pour le système OPALE:



Ce schéma correspond à la recherche d'une solution. La solution fournie par le système AGATE est la première possible dans la recherche générale non-déterministe. Cette recherche à partir d'une multi-étiquette suit un schéma similaire à celui utilisé par le système OPALE:



Comme pour le langage OPALE, le langage AGATE se décompose en quatres parties:

- Définition de l'environnement.
- Définition de la liste des règles initiales.
- Définition de l'adressage des règles.
- Définition des règles.

La syntaxe et la sémantique sont identiques à celles du système OPALE. Seules les origines des variables sont différentes:

EC : étiquette courante en cours d'évaluation.

ED : étiquette issue de la lecture du dictionnaire AGATE.

EI : étiquette initiale du traitement.

EP$_i$: i compris entre 1 et 10, étiquettes analysées précédemment par le système AGATE (cette position étant relative à l'étiquette courante, elle varie à chaque appel du système AGATE. Elle est comptée de droite à gauche).

EP$_i$* : i compris entre 1 et 10, étiquettes analysées précédemment par le système AGATE; la position relative ne compte que si l'étiquette concernée a fourni une chaîne non-vide. Dans le cas contraire, la position correspondante est sautée. Cette position ne peut dépasser les dix positions absolues (E$_i$) et par conséquent peut ne pas être définie.

- Dans le cas ou une étiquette n'est pas définie (EPi ou EPi*) elle est identifiée à l'étiquette vide (toutes ses valeurs nulles).
- Les seules étiquettes affectables sont EC, ED et EI.

Pour le traitement ou le test des chaînes, les définitions sont similaires à celles du système OPALE:

CC : chaîne courante en cours d'élaboration.

CP$_i$: i compris entre 1 et 10, chaînes provenant de l'analyse de la ieme étiquette précédente analysée (chaîne provenant de l'analyse de l'étiquette E$_i$).

CP$_i$* : i compris entre 1 et 10, chaînes provenant de l'analyse de la ieme étiquette précédente ayant fourni une chaîne non-vide (chaîne provenant de l'analyse de l'étiquette E$_i$*).

Les fonctions spécifiques pour le système AGATE sont les suivantes:

TCHAINE : altération de la chaîne en cours d'élaboration. C'est la seule fonction permettant de définir un résultat de l'analyse. Cette fonction a quatre paramètres. Les trois derniers ont la même signification que ceux de la fonction TCHAINE du système OPALE et le premier détermine la chaîne à modifier (Dans le système OPALE, seule la chaîne courante CC peut être modifiée). Les chaînes modifiables dans le système AGATE sont toutes les chaînes manipulables: CC, CP$_i$, et CP$_i$*.

HALT : arrêt du système (solution). Sans cette fonction, le système AGATE ne s'arrêterait que lorsque la lecture du dictionnaire ne donnerait pas de résultat. Dans ce dernier cas, cela correspondrait au balayage de toutes les solutions et ne donnerait aucun résultat. L'application d'une règle contenant la fonction HALT définit une solution qui est par convention la solution du système AGATE. Il est à noter que, dans le cas ou cette règle comporte des sous-règles, l'application de cette fonction bloque la recherche des sous-règles qui deviennent ainsi inopérantes, la solution étant déterminée juste après l'application de cette règle (après les affectations et les fonctions TCHAINE présentes). Dans le cas où les fonctions SNUL et HALT sont toutes les deux présentes la fonction HALT a priorité et annule cette fonction SNUL.

SNUL : fonction similaire à la fonction HALT. La solution est ici définie par la chaîne vide (solution nulle ou génération du mot vide).

NONSEP : élimination du séparateur entre deux mots. Le mot produit par le système AGATE sera accolé au mot précédent.

ASEP : Définition d'un séparateur. Le mot produit par le système AGATE sera séparé du mot précédent par un espace. Ce fonctionnement correspond au fonctionnement par défaut.

NONLIG : Le passage à la ligne automatique est supprimé. Une ligne du fichier RESUL peut avoir une dimension quelconque.

ALIGN : Le passage à la ligne automatique est établi. Ceci correspond au fonctionnement normal du système. La valeur retenue pour cette fonction (NONLIG ou ALIGN) correspond à celle appliquée en dernier lors du traitement complet du texte.

LCTFORM : lecture d'un format. Cette fonction a comme paramètre un nom de format qui doit être affectable par une procédure d'entrée/sortie. Le traitement de cette fonction positionne le format paramètre avec le contenu du fichier d'entrée des formats. Chaque appel lit un format dans l'ordre du fichier. Le nom du fichier de lecture de formats est par défaut MODINT/FORMATS.sygeft, il peut être redéfini dans le fichier paramètre.

ECRFORM : écriture d'un format. Cette fonction a comme paramètre un nom de format qui doit être affectable par une procédure d'entrée/sortie. Le traitement de cette fonction complète le contenu du fichier de sortie des formats avec les valeurs du format paramètre. Le nom du fichier d'écriture de formats est par défaut MODINT/FORMATS.sygsft, il peut être redéfini dans le fichier paramètre.

LCTMESSG : lecture d'une variable chaîne d'un format. Cette fonction a comme paramètre un nom de format qui doit être affectable par une procédure d'entrée/sortie et une variable chaîne de ce format. Le traitement de cette fonction positionne la variable chaîne de ce format paramètre avec le contenu d'un fichier identifié comme fichier d'entrée des messages. Le nom du fichier de lecture de message est par défaut MESSAGE.sygent, il peut être redéfini dans le fichier paramètre.

ECRMESSG : écriture d'une variable chaîne d'un format. Cette fonction a comme paramètre un nom de format qui doit être affectable par une procédure d'entrée/sortie et une variable chaîne de ce format. Le traitement de cette fonction complète le contenu d'un fichier, identifié comme fichier de sortie des messages, avec la variable chaîne de ce format paramètre. Le nom du fichier d'écriture de message est par défaut MESSAGE.sygsrt, il peut être redéfini dans le fichier paramètre.

LCTMESSGL : lecture d'une variable chaîne d'un format. Cette fonction a comme paramètre un nom de format qui doit être affectable par une procédure d'entrée/sortie et une variable chaîne de ce format. Le traitement de cette fonction positionne la variable chaîne de ce format paramètre avec

le contenu d'un fichier identifié comme fichier d'entrée des messages. La différence avec la fonction LCTMESSG vient du mode de lecture. La fonction LCTMESSG lit le fichier complètement et tronque éventuellement son contenu. Cette fonction, LCTMESSGL, par contre ne lit qu'une seule ligne à chaque appel, le caractère fin de ligne étant supprimé. Chaque appel de cette fonction lit donc les lignes du fichiers des messages) les unes après les autres. Les deux fonctions traite le même fichier.

Chapitre 5

Les dictionnaires

Les dictionnaires contiennent la base de connaissance du système. Ils sont de trois types différents: Le dictionnaire de formats contient des définitions standards de configurations d'étiquettes; Le dictionnaire de chaînes définit les segments nécessaires à l'analyse du texte par le système OPALE; Les dictionnaires d'étiquettes définissent un ensemble d'étiquettes à associer à une configuration donnée.

5.1 Le dictionnaire des formats

Pour une définition donnée il peut être utile d'avoir un ensemble de valeurs de variables définies à l'avance (utilisation fréquente d'un groupe de variables affectées des mêmes valeurs). Dans ce cas on peut définir des profils de variables affectées ou format. La référence à ce format peut se faire dans tous les sous-systèmes et tous les langages (dictionnaires ou grammaires).

Un ensemble de formats est défini par rapport à une définition. Chaque format de cet ensemble comporte un nom (par rapport à une même définition de référence tous les noms de formats doivent être distincts). Le nom de format est suivi de la liste des variables non vide de ce format affectées de leurs valeurs. Cette liste d'affectation de variables ne concerne que des constantes ou des variables de formats préalablement définies. Chaque variable affectée est non localisée.

Pour une définition la liste des formats commence par le symbole:

&REFER(<nom de définition>).

Le dictionnaire des formats à la forme générale suivante:

&REFER(<nom de définition>).

<nom de format 1> : suite d'affectations .
<nom de format 2> : suite d'affectations .
.....
.....
.....

&REFER(<nom de définition>).

.....
.....
.....

Il peut exister plusieurs listes de formats pour la même définition. Ces listes sont alors considérées comme concaténées pour n'en former qu'une seule.

Un format est affectable. C'est à dire que sa valeur peut évoluer lors de l'application quelconque d'une compilation ou d'une exécution. De ce fait un format peut représenter une étiquette constante repérable

dans l'application d'une exécution ou d'une compilation. Lorsqu'un format est modifié par une compilation ou une exécution il conserve sa valeur jusqu'à la fin de cette compilation ou exécution, ou bien sûr jusqu'à une nouvelle affectation. Une variable chaîne ou externe d'un format ne pourra jamais être plus longue que la chaîne initiale définie lors de la création de ce format (les fonctions %ESPACE et %NUL permettent de réserver des zones de longueur quelconque). Pour l'utilisation d'un format comme registre il est donc souvent nécessaire de l'initialiser par une affectation située dans une grammmaire.

Il est possible de définir des formats non affectables. Ces formats suivent le mot-clé "&CONSTANTES.". A l'intérieur d'une suite de formats constants il est possible de revenir à la définition de formats affectables par le mot-clé "&REGISTRES.".

La définition des formats affectables par les procédures d'entrée/sortie doivent suivre le mot clé "&REGCOM". Dans ce cas la longueur maximum des variables chaînes ou externes est celle définie par le système pour chacun de ces types. Après la définition des ces formats d'entrée/sortie il est possible de revenir à la définition des formats constants ou affectables avec les mots-clés correspondants.

Ce ditionnaire peut contenir la définition d'une matrice. Cette matrice pourra alors être utilisée par une fonction "VECTEUR". La définition d'une matrice s'effectue ligne à ligne. L'ensemble des lignes est encadré par les mots clés "&MATRICE:" et "&FINMAT.". L'en-tête contient la description de le nombre d'éléments d'une ligne et son interprétation: La saisie des éléments d'une ligne pourra être incomplète (les éléments non décrits seront nuls) et dans un type différents. Cette ligne pourra également être normalisée aprés la saisie (La somme des carrés de tous ces éléments sera égale à 1. Les en-tête auront donc la forme :

- &MATRICE: <nom mat>(<type saisie>,<nombre éléments ligne>)->(<type resultat>,NORM).
- &MATRICE: <nom mat>(<type saisie>,<nombre éléments ligne>)->(<type resultat>).

La description des lignes sera définie par une suite d'éléments séparés par des virgules entre les sympoles ':' et '.'. Chaque élément est une constante et suit les règles d'écriture de son type. Un commentaire (par exemple le numéro de ligne peut précéder le symbole de départ d'une ligne ':'.

Par exemple la matrice suivante sera une matrice de nombres flottants, chaque composante non nulle sera égale et la ligne sera normalisée:

&MATRICE: transfo(ARITH,880)->(FLOTTANT,NORM).

1: 0,0,1,1,1,0,1.

2: 1,0,1.

&FINMAT.

5.2 Le dictionnaire de chaînes

Ce dictionnaire est nécessaire au système OPALE. Ce dernier ne fait référence au cours d'une exécution qu'à un seul dictionnaire de chaînes. Ce dictionnaire est structuré et comporte donc un adressage. L'adressage de ce dictionnaire s'effectue à deux niveaux: un premier niveau arborescent qui partage le dictionnaire en plusieurs sous-dictionnaires et un niveau terminal qui est constitué d'un ensemble de segments associés de valeurs.

L'adressage arborescent du dictionnaire utilise un ensemble de clés définies par des valeurs de variables. Une partie de dictionnaire est adressée à partir de la valeur d'une variable contenue dans l'étiquette qui définit l'adressage de ce dictionnaire (étiquette EC du système OPALE). Le système OPALE étant non deterministe plusieurs parties du dictionnaire peuvent être adressées par cette étiquette. La multiplicité

de l'adressage dépend de la nature de la clé définissant les sous parties du dictionnaire. Deux types de clés sont donc définies: le type exclusif et le type non exclusif. Le type non-exclusif induit une consultation et éventuellement un résultat multiple. Ce type de clé suppose naturellement une variable non-exclusive et dans ce cas toutes les parties du dictionnaire qui seront définies par une valeur contenue dans la variable index seront consultées. Dans le cas d'une clé exclusive seule la partie qui correspond à la valeur de la clé est consultée. Une clé exclusive peut être définie par une variable arithmetique, exclusive, potentielle ou même non-exclusive. Dans ce dernier cas il est important de noter que la valeur exclusive de cette variable correspond à un profil précis de la variable (sous ensemble de l'ensemble des valeurs possibles). La constitution de cette valeur exclusive peut donc nécessiter une union de valeur.

Exemple:

Si la variable EXP est une variable non exclusive avec les valeurs A,B,C (définition: EXP(A,B,C)), et si cette variable définit une clé exclusive les valeurs de cette clé pourront être:

- A - B - C - A|B - A|C - B|C - A|B|C

Les symboles de définition de cette arborescence d'adressage sont:

&CLEX(<nom de variable>). définition d'une clé exclusive.

&CLNEX(<nom de variable>). définition d'une clé non-exclusive.

&VAL(<nom de variable> , <valeur de variable>). définition d'un niveau ou sous-arborescence qui correspond à la clé ayant la même variable. Comme l'arborescence est définie par des blocs imbriqués le niveau concerné est le dernier défini dans le cas où plusieurs niveaux feraient référence à la même variable. Ce niveau sera consulté si la variable de l'étiquette de référence aura cette valeur au cours de l'appel.

&MORPHE. définition des segments.

Ce mot clé introduit une partie terminale du dictionnaire de chaînes. Cette partie est composée d'une suite de segments associés de valeurs:

'segment' : valeur d'étiquette.
.....
.....
'segment' : valeur d'étiquette.

&FIN. fin du dictionnaire. Ce mot clé est facultatif pour le dictionnaire de chaînes. Dans le cas où il est présent, il termine le dictionnaire.

La valeur d'étiquette est définie par une suite d'affectations de valeurs de variables. Elle est associée à chaque entrée du dictionnaire (segment défini sur une feuille de l'arborescence d'adressage). Cette suite d'affectations correspond à l'étiquette lue dans le dictionnaire. Pour définir cette suite d'affectations on utilise soit des variables non localisees soit des variables formats. Les variables affectées sont toutes non localisées et sont du type chaîne, arithmetique, exclusif, non exclusif ou potentiel. La chaîne d'entrée (segment) est placée entre apostrophes; si un segment doit contenir une apostrophe cette dernière doit être doublée. Plusieurs segments identiques peuvent apparaître dans cette partie de dictionnaire. Cette possibilité constitue la deuxième méthode pour obtenir des solutions multiples. Toutes les lectures possibles de ces segments seront en effet examinées.

La lecture du dictionnaire est conditionnelle. La partie lue correspond à celle définie par la valeur des variables définies dans l'adressage. La forme minimum du dictionnaire correspond à un seul ensemble de segments introduits par le mot clé "&MORPHE.". Dans ce cas cet ensemble est toujours consulté lors de la lecture du dictionnaire:

&MORPHE.

'segment' : valeur.
.....
.....
'segment' : valeur.

Dans le cas où un adressage à été défini, les parties consultées correspondent toujours à celles définies par la clé d'adressage:

```
&CLEX(exemple).

    &VAL(exemple,1).
    &MORPHE.

        'segment' : valeur.
                .... partie consultée
                .... si la variable exemple
                .... a la valeur 1.
        'segment' : valeur.

    &VAL(exemple,2).
    &MORPHE.

        'segment' : valeur.
                .... partie consultée
                .... si la variable exemple
                .... a la valeur 2.
        'segment' : valeur.
```

5.3 les dictionnaires d'étiquettes

Un dictionnaire d'étiquettes peut être défini pour chacun des systèmes OPALE, TELESI ou AGATE. Ce dictionnaire est bien sûr obligatoire dans le système AGATE et facultatif dans les autres systèmes. Un dictionnaire d'étiquettes définit un ensemble d'étiquettes associées à une configuration de variables. Cette configuration de variables détermine le mode d'adressage qui est arborescent comme dans le cas du dictionnaire de chaînes. Seul l'adressage du dernier élément (feuilles) est différent pour ce dictionnaire et doit être défini par une clé exclusive. De plus pour chaque niveau de ce dictionnaire une valeur de clé particuliere peut être définie et correspond au cas où la consultation de ce niveau ne définit pas d'adressage.

Le dictionnaire d'étiquette peut comporter des références croisées au niveau des blocs terminaux. Pour cela les définitions de ces niveaux sont étiquetables par un nom. Ce nom permettra de définir une référence utilisable pour un bloc associé à une valeur.

Les mots clés de définition de ce dictionnaire sont les suivants:

&CLEX(<nom de variable>). définition d'une clé exclusive.

&CLNEX(<nom de variable>). définition d'une clé non exclusive. La lecture d'un dictionnaire d'étiquettes est en général unique (lecture du dictionnaire dans une expression) et ne donne par consequent qu'un seul résultat. Dans le cas de l'utilisation d'une clé non exclusive les parties du dictionnaire seront consultées dans l'ordre où elles ont été définies et la première lecture satisfaisante définira cette consultation.

&VAL(<nom de variable> , <valeur>). définition de la sous arborescence d'adressage (niveau). Même signification et contrainte que pour le dictionnaire de chaînes.

&SINON(<nom de variable>). définition de la consultation par defaut (clé correspondante ne définissant pas de sous-parties). Cette partie terminale est constituée d'une suite d'éléments composés d'une valeur de variable de la clé suivie d'une valeur d'étiquette:

&VAL. définition de la sous-partie terminale de l'arborescence de consultation. Le clé correspondante doit être exclusive.

```
    &VAL.
```

<valeur 1> : valeur étiquette.
<valeur 2> : valeur étiquette.
.....
.....
<valeur n> : valeur étiquette.

Pour une même valeur d'étiquette il est possible de définir plusieurs étiquettes associées. L'accès à une de ces étiquette est alors définie par un index (variable dictionnaire indexé). L'index représente le numéro d'étiquette à considérer. Le système ne vérifie pas que la valeur de l'index soit acceptable. Dans le cas ou la valeur de l'index est plus importante que le nombre d'étiquettes définies pour une valeur de la clé le résultat de la lecture du dictionnaire est aléatoire. Les étiquettes associées à une même valeur d'étiquette sont indexées de 1 à n. La valeur 1 de l'index est équivalente à son absence et définit la première étiquette. Toutes les étiquettes associées à une même valeur d'index sont ordonnées. Elles suivent la première définition et sont identifiées par des guillemets:

<valeur 1> : valeur étiquette. (index 1 de la valeur 1)
" : valeur étiquette. (index 2 de la valeur 1)
.....
" : valeur étiquette. (index n de la valeur 1)
<valeur 2> : valeur étiquette. (index 1 de la valeur 2)
" : valeur étiquette. (index 2 de la valeur 2)
.....
" : valeur étiquette. (index n de la valeur 2)
.....
<valeur p> : valeur étiquette. (index 1 de la valeur p)
" : valeur étiquette. (index 2 de la valeur p)
.....
" : valeur étiquette. (index n de la valeur p)

&VALCP. deuxième définition de la sous-partie terminale de l'arborescence de consultation. La clé correspondante doit également être exclusive. Cet adressage est identique au précédent mais la valeur associée à une entrée est initialisée à la valeur de l'index. Ainsi les deux définitions suivantes sont équivalentes:

&CLEX(ul).
&VAL.
<valeur 1> : ul = <valeur 1>; valeur étiquette.
<valeur 2> : ul = <valeur 2>; valeur étiquette.
.....
<valeur n> : ul = <valeur n>; valeur étiquette.

&CLEX(ul).
&VALCP.
<valeur 1> : valeur étiquette.
<valeur 2> : valeur étiquette.
.....
<valeur n> : valeur étiquette.

&VALCD. troisième définition de la sous-partie terminale de l'arborescence de consultation. La clé correspondante doit également être exclusive. Dans cette définition terminale chaque valeur définit une liste de conditions qui seront testées dans l'ordre indiqué. La valeur résultante sera celle associée à la première condition vraie ou égale à l'étiquette nulle dans le cas où aucune condition de la liste n'est vérifiée. La suite des conditions et leurs valeurs associées sont placées entre accolades:

&VALCD.
VAL1 { COND1 : valeur étiquette.
COND2 : valeur étiquette.
...
...
CONDN : valeur étiquette.
}

Bien sûr pour une même condition il est possible de définir plusieurs valeurs d'étiquettes. Comme pour le cas précédent les différentes étiquettes sont ordonnées et définies par des guillemets. Ces différentes valeurs sont accessibles par des variables indexées.

&SELECTD(<var sel>). Définition d'une clé de sélection. Cette clé définit une sous-partie terminale. avec comme clé exclusive la variable de sélection. Cette variable ne sera pas prise en compte pour une consultation dicrecte. Comme cette partie est référencable, cette variable pourra servir de clé exclusive pour un autre adressage. Cette clé doit obligatoirement être suivie d'une définition spécifique d'une partie terminale (&VALCP( )). Toutes les entrées sélectionnées à partir de cette clé seront candidates pour la sélection. Cette sélection s'effectuera par la recherche d'un minimum ou d'un maximum.La recherche par dédaut est celle du minimum. La définition de cette clé peut également définir le prédicat (minimum ou maximum) qui doit être utilisé lors de la sélection. La forme est alors:
&SELECTD(<var sel>):MIN. ou &SELECTD(<var sel>):MAX.

&VALCP(<var calcul>,<procédure>). Définition d'une partie terminale d'une clé de sélection. Cette forme ne peut suivre que la définition "&SELECTD". La consultation s'effectue par l'évaluation de la procédure définie par cette clé. L'entrée retenue sera celle qui minimise (ou maximise) la valeur de la variable "calcul". La procédure d'affectation définie une étiquette temporaire. La variable "calcul" considérée sera celle de cette étiquette. Pour l'elaboration de cette étiquette, les étiquettes accessibles sont EX: étiquette correspondant à la valeur de l'entrée du dictionnaire, EQ: étiquette de consultation et "*": étiquette en cours d'évaluation.

&SELECTX(<var sel>). Définition d'une clé de sélection par référence croisée. Cette clé est aussi une clé de selection (minimisation d'une valeur évaluée au cours de la consultation). Elle est suivie obligatoirement par une forme spécifique d'une sous partie terminale. Cette sous-partie fait référence à une partie terminale définie dans une autre branche d'adressage. Cette forme peut également définir le prédicat utilisé pour la sélection de la même manière que précédement.

&VAL(<var calcul>,<procédure>): <nom référence>. Définition d'une partie terminale de selection par référence croisée. La partie de dictionnaire référencée par <nom> doit être une partie terminale. La clé de sélection peut être remplacée par '*'. La lecture de cette partie de dictionnaire s'effectue par la partie index et par le contenu des éléments eux-mêmes. Dans un premier temps la clé de sélection permet de définir un sous ensemble de la partie de dictionnaire repérée. Si la clé de sélection est '*' toute la partie repérée est sélectionnée. Ensuite, comme pour toute sélection, la procédure d'affectation est évaluée pour chaque élément de l'ensemble ainsi défini. La valeur de la variable de calcul désignée par la clé est consultée après chaque évaluation. L'entrée adressée sera celle correspondant à la valeur minimale (ou maximale) de cette variable.

&SINON: définition de la consultation par defaut dans le cas de l'adressage terminal non conditionnel.

&SINON{ définition de la consultation par defaut dans le cas de l'adressage terminal conditionnel. La liste des conditions et de leurs valeurs associées se termine par une accolade fermante:

```
&SINON{ COND1 : valeur étiquette.
        COND2 : valeur étiquette.
          ...
          ...
        CONDN : valeur étiquette.
      }
```

Définition croisée Une partie du terminale du dictionnaire peut être étiquetée. Cette partie pourra alors être accessible de plusieurs façons:

<nom>: &VAL. bloc simple.
<nom>: &VALCP. bloc avec initialisation.
<nom>: &VALCP(...) bloc de définition de sélection.
<nom>: &VALCD. bloc conditionnel.
&VAL: <nom>. définition de l'identification d'un bloc comme étant celle définie par <nom>.

Cette construction peut avantageusement être utilisée pour définir des sous conditions.

Ainsi si on définit une première clé par une variable agrégat comportant les variables v1 et v2 (va = vgb(v1,v2)). nous pouvons avoir la structure suivante:

```
&CLEX(exp).
    &VAL(exp,1).
    &CLEX(va).
      repère: &VAL.
            ....
            ....
    &VAL(exp,2).
    &CLEX(v1).
      &VAL: repère.
    &VAL(exp,3).
    &CLEX(v2).
      &VAL: repère.
```

Si la variable exp vaut 1 la lecture sera conditionnée par la valeur de v1 et de v2;
Si cette variable vaut 2 seule la valeur de v1 déterminera l'accès et
Si la valeur est 3 seule celle de v2 sera prise en compte.

&FIN. fin du dictionnaire d'étiquettes.

Une variable arithmetique peut être calculée automatiquement par le compilateur pour chaque clé exclusive. Cette variable associée à une clé exclusive doit appartenir à la même variable globale que la variable définissant cette clé. Cette variable sera affectée d'un numéro d'ordre différent pour chaque clé identique. Ce numéro d'ordre commencera à la valeur 0. La forme syntaxique est la suivante:

&CLEX(<nom de variable>,<nom de variable arithmétique>).

<nom>: &CLEX(<nom de variable>,<nom de variable arithmétique>).

Lorsqu'une clé exclusive est définie par une variable agrégat la définition des valeurs de cette clé suppose souvent une répétition du nom des variables internes. Il est possible de définir par avance cette répétition dans la définition de la clé. Le nom de la variable globale doit alors être suivi de la liste des variables à affecter:

&CLEX(<nom variable agrégat> [liste des noms de variables]).

Exemple:

Soit la variable agrégat COMP contenant deux variables potentielles UL1 et UL2. La definition des valeurs pourra alors prendre l'une des deux formes suivantes:

premier cas:

```
&CLEX(COMP).
  &VAL.
        UL1− >'V1' | UL2− >'V2' : ..... .
        UL1− >'V11' | UL2− >'V21' : ..... .
```

deuxième cas:

```
&CLEX(COMP[UL1,UL2]).
  &VAL.
        'V1' 'V2' : ..... .
        'V11' 'V21' : ..... .
```

Lorsqu'une clé exclusive est constituée d'une variable externe il est nécessaire de préciser alors le type d'élément composant le vecteur: ARITH, ARITHL, FLOAT ou DOUBLE. On précise la nature des éléments composant le vecteur après le nom de la variable:

&CLEX(<nom variable externe> : <nature des éléments>).

Pour la définition d'une clé les différentes composantes du vecteur sont alors séparées par des virgules et leurs formes correspond à celle d'une constante entière ou flottante suivant le cas.

&CLEX(Vecteur : ARITH).
&VAL.

12,24,48,...,4 : valeur étiquette.
8,16,.....,32,3 : valeur étiquette.
.....
.....
16,3,4,19,...,10 : valeur étiquette.

Lorsqu'une clé est associée à une variable de type chaîne la reconnaissance d'une entrée pourra concernée les préfixes: une entrée sera recounnue si la chaîne candidate est un préfixe de la chaine index du dictionnaire. Pour préciser ce cas, le nom de la variable chaîne est suivit de la forme: [~*]

&CLEX(chaine[~*]).

Ce dictionnaire peut contenir également des procédures. Les procédures conditionnelles pourront apparaître à l'intérieur des conditions de clés conditionnelles. Les procédures d'affectation permettrons de définir les valeurs de références pour l'adressage sélectif. Toutes ces procédures doivent être déclarées en tête de fichier avant l'adressage arborescent du dictionnaire et derrière les variables agrégats éventuelles. Les procédures conditionnelles sont introduites par le mot clé "&COND." et les procédures d'affectation par "&AFCT.". Dans les procédures d'affectation les variables affectées sont non localisées. Les variables définissant les expresson sont elles localisées:

* variable de l'étiquette en cours d'évaluation.

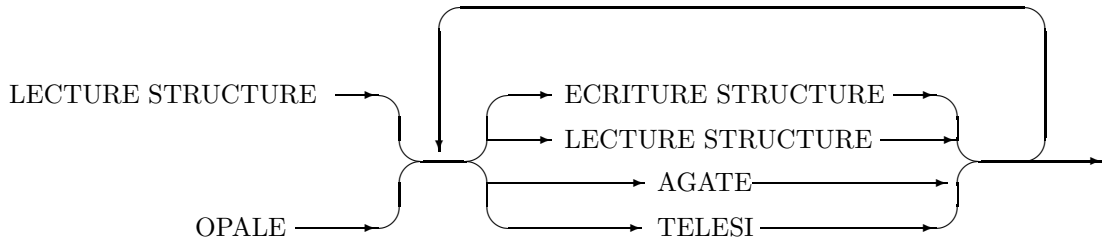
EX variable de l'étiquette correspondant à l'entrée traitée du dictionnaire.

EQ variable de l'étiquette consultée.

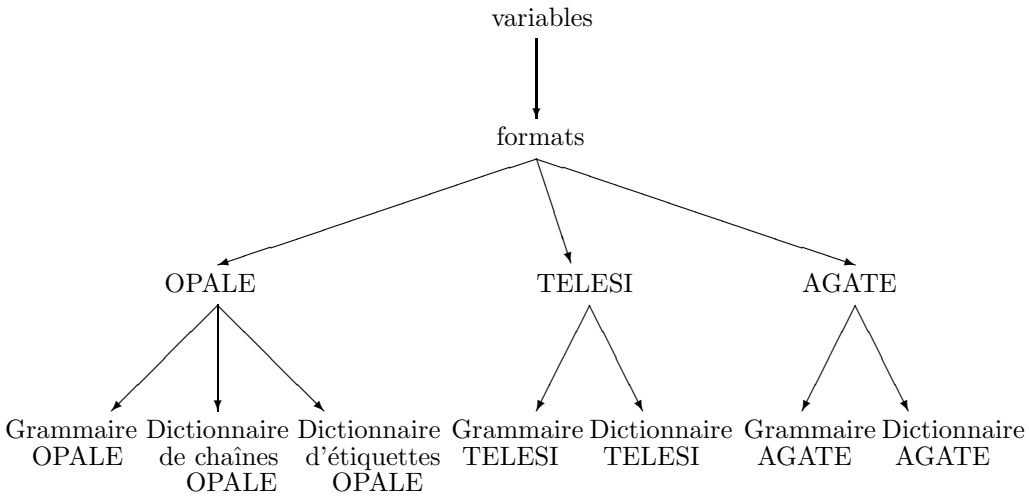
Chapitre 6

Mise en oeuvre

Une application complète du système correspond au traitement d'un texte. Ce traitement restitue un texte déduit du texte source par l'application des différents systèmes OPALE, TELESI, et AGATE. Ce traitement suit un schéma d'états finis:



La description des différents appels s'effectue dans un fichier d'appels et doivent correspondre au parcours de cet automate. Chaque sous-système simule une grammaire précédemment compilée et qui se trouve dans un fichier pouvant contenir un nombre quelconque de grammaires. Ces systèmes peuvent faire appel également à des dictionnaires compilés eux aussi avant le traitement. Certains dictionnaires sont facultatifs (la grammaire simulée ne fait pas référence à des renseignements situés dans un dictionnaire) ou obligatoires (sous-systèmes OPALE et AGATE). Au total pour fonctionner le système a besoin de neuf fichiers compilés dont la dépendance est résumé dans le schéma suivant:



Lorsque l'appel du système s'effectue que sur un sous système les fichiers des autres sous-systèmes sont facultatifs.

La mise en oeuvre comporte deux phases: la première concerne la compilation des différents langages des sous-systèmes; la deuxième l'exécution du système sur un texte conformément aux différentes grammaires

compilées précédemment.

Cette mise en oeuvre doit respecter la convention suivante: Le répertoire de travail à partir duquel sont lancés tous les appels de programmes contient au moins deux éléments:

- Un répertoire de nom "MODINT"
- Un fichier (ou un lien sous UNIX) de messages (par exemple MSGR)

Les résultats des compilations seront placés dans ce répertoire MODINT.

6.1 Les compilations

Chaque compilation de grammaire ou de dictionnaire sont indépendantes les unes des autres. Par contre toutes ces compilations dépendent de la compilation des variables et des formats. Une modification et une recompilation de l'un de ces deux derniers fichiers nécessitent par conséquent une recompilation de tous les autres éléments.

Chaque compilation a la forme:

<nom du compilateur> <nom du fichier de messages> <nom du fichier source> [<paramètres>]

Les différents appels sont les suivants:

compvar compilation des variables.

compfor compilation des formats.

compdch compilation des dictionnaires de chaînes.

compdctq compilation des dictionnaires d'étiquettes. Dans ce cas l'appel nécessite trois paramètres et a la forme :
compdctq <nom du fichier de messages> X <fichier source> [<paramètres>]
le caractère X précise le dictionnaire d'étiquettes concerné par cet appel. il peut être O, T ou A suivant que l'on compile un dictionnaire pour le système OPALE AGATE ou TELESI.

compgro compilation de la grammaire OPALE.

compgrtl compilation d'une grammaire TELESI.

compgra compilation d'une grammaire AGATE.

Les paramètres définissent la forme du listing et réalisent les fonctions d'initialisation. Les paramètres communs à tous les compilateurs sont:

- -i Le fichier contenant les résultats des compilations (associé à ce compilateur) sera initialisé. Ce paramètre est bien sûr obligatoire lors de la première compilation.
- -c Le listing de sortie sera défini par le fichier standard de sortie (écran). Si ce paramètre n'est pas présent un fichier "SORTIE" sera créé. Une compilation qui utilise un fichier "SORTIE" détruit toujours le fichier "SORTIE" existant.
- -Ln Ce paramètre est exclusif du précédent. Il précise le nombre de lignes par page du fichier "SORTIE".
- -n Ce paramètre annule le résultat de la compilation. Ce dernier ne sera pas rangé dans le fichier approprié. Ce fichier ne sera donc pas mis à jour mais il pourra être quand même être initialisé si le paramètre -i est présent.

Les paramètres spécifiques concernent la compilation du dictionnaire de chaîne et la compilation de la grammaire TELESI.

Paramètre spécifique à la compilation du dictionnaire de chaine:

- -d Le découpage en segment par le systéme OPALE peut s'effectuer dans le sens gauche droite ou droite gauche. Le dictionnaire de chaine doit être compilé en fonction du sens qui sera utilisé. Sans paramètre spécifique le compilateur suppose que le sens gauche droite sera utilisé. Avec la présence de ce paramètre le sens droite gauche sera pris en compte.

- -r La compilation du dictionnaire correspond à un complément ou une mise à jour. Le dictionnaire de référence sera complété ou mis à jour avec le contenu de ce fichier. Ce complément s'effectuera en accord avec l'adressage des index et correspondra toujours à une substitution de zone complète correspondant aux zones de plus haut niveau du dictionnaire de mise à jour qui doivent correspondre à des zones existantes ou potentielle dans le dictionnaire existant. La position choisie pour le remplacement ou le complément est toujours la plus haute et la plus à gauche possible. Dans le cas ou l'index de ce dictionnaire est incompatible avec celui du dictionnaire à mettre à jour la compilation est sans effet. Par contre si le dictionnaire à mettre à jour n'existe pas il est créé avec le contenu de ce fichier.

Paramètre spécifique à la compilation de la grammaire TELESI:

- -O Optimisation de la lecture du dictionnaire d'étiquettes. Cette optimisation concerne uniquement la lecture du dictionnaire. Lorsqu'une lecture est effectuée la référence à cette lecture est concervée le plus longtemps possible, évitant ainsi certaine relecture.

- -D Avec ce paramètre le listing de compilation comportera après chaque règle l'interprétation de chaque transformation. Cette option permet de vérifier que la génération des listes est bien celle attendue.

6.2 L'exécution

L'execution du système sur un texte nécessite trois éléments. Le premier élément est constitué par le fichier des messages, le deuxième par le fichier des paramètres d'appels des systèmes et le troisième par le fichier contenant le texte à traiter. Les paramètres d'appels peuvent contenir ces éléments ainsi que des caractéristiques associés aux traitement.

L'exécution produit éventuellement les fichiers suivants:

- Le fichier "LISTE" qui contient le texte source.
- Le fichier "RESUL" qui contient le texte résultat.
- Le fichier "SORTIE" qui contient les visualisations intermédiaires.
- Le fichier "<ficher texte>.<ext>".

La présence ou non de ces fichiers dépend de la valeur des paramètres.

Le fichier paramètres d'appels contient une suite de blocs de commande, chaque bloc correspondant à une transition dans l'automate défini précédemment. Une transition est un appel à un sous-système ou la réalisation d'une fonction. Aussi chaque bloc débute par son identification constituée du nom du sous-système ou de la fonction concernée. Nous trouvons toujours dans ces paramètres de bloc les définitions des grammaires et dictionnaires à utiliser , les modes de traitement et les visualisations des résultats d'application de sous-systèmes.

La structure des différents blocs est la suivante:

bloc OPALE :

'OPALE'
'<nom de grammaire>' '<nom de dictionnaire de chaîne>'
<nom de dictionnaire d'étiquette>'
'<nom de définition concernee>'
'X' visualisation du fonctionnement de l'automate.

X=1 visualisation, X=0 sans visualisation.

'X' forme de la visualisation.

X=1 visualisation detaillee, X=0 visualisation rapide.

'X' visualisation du résultat du système OPALE

X=1 visualisation, X=0 sans visualisation.

'X' forme de la visualisation du résultat.

X=1 visualisation du résultat detaillee, X=0 visualisation rapide.

bloc LECTURE STRUCTURE :

'LECTURE STRUCTURE'

'<nom de référence>'

Lecture de l'élément structuré préalablement rangé sous cette référence. Le nom de référence correspond à un nom de fichier relatif au répertoire "MODINT". Dans le nom de référence, le caractère "%" sera éliminé, le caractère "*" sera remplacé par le nom du fichier traité, et le caractère "#" par le nom du fichier sans l'extension. Si le nom du fichier n'existe pas le nom par défaut sera utilisé ("MODINT/FORMATS.sygeft").

bloc TELESI :

'TELESI'

'<nom de grammaire>'

'<nom de dictionnaire d'étiquette>'

'<nom de définition concernee>'

'<nom de la grammaire d'entrée utilisee>'

'<nom de la variable globale>'

'<nom de la variable>'

Ces deux derniers paramètres permettent de définir la variable qui sera utilisée dans les visualisations. La valeur de cette variable sera associée à chaque point des arborescences d'un élément structuré visualisé.

nombre de dimensions de l'élément structuré.

'X' '<nom de grammaire de départ>' '<nom de grammaire de fin>'

ce paramètre définit la visualisation de l'exécution. Si X=0 il n'y a pas de visualisation. Si X=1 il y aura visualisation entre la grammaire de départ et la grammaire de fin (comprises). Si un nom est absent cela revient à visualiser à partir de la première grammaire ou à finir cette visualisation après la dernière grammaire.

'X' '<nom de grammaire de départ>' '<nom de grammaire de fin>'

ce paramètre concerne le mode de visualisation (détaillé ou non) et a les mêmes significations que précédemment.

'X' visualisation du résultat du système TELESI.

'X' forme de la visualisation.

Ces deux derniers paramètres ont une signification identique à ceux définis pour le système OPALE.

'<nom grammaire>' '<nom dictionnaire de chaîne>' '<nom dict. d'étiq>' 'X' 'X'

Ces cinq derniers paramètres définissent les conditions d'un appel au système OPALE par le système TELESI. les deux booléens concernent dans l'ordre la visualisation et le mode de visualisation.

bloc ECRITURE STRUCTURE :

'ECRITURE STRUCTURE'

'<nom de référence>

Ce nom identifiera l'élément structuré qui sera rangé et qui pourra par conséquent être utilisé pour des appels ultérieurs. Le nom de référence correspond à un nom de fichier relatif au répertoire "MODINT". Dans le nom de référence, le caractère "%" sera éliminé, le caractère "*" sera remplacé par le nom du fichier traité, et le caractère "#" par le nom du fichier sans l'extension. Si le nom du fichier n'existe pas le nom "MODINT/FORMATS.sygeft" sera utilisé.

bloc LECTURE ETIQUETTES NOMMEES :
'LECTURE ETIQUETTES NOMMEES'
'<Type de lecture>'

- FUSION: les étiquettes lues complètent les étiquettes existantes.
- REMPLACEMENT: les étiquettes lues remplaceront les étiquettes existantes.

'<nom de référence>'

Lecture des étiquettes nommées préalablement rangées sous cette référence. Le nom de référence correspond à un nom de fichier relatif au répertoire "MODINT". Dans le nom de référence, le caractère "%" sera éliminé, le caractère "*" sera remplacé par le nom du fichier traité, et le caractère "#" par le nom du fichier sans l'extension. Si le nom du fichier n'existe pas le nom "MODINT/FORMATS.sygeft" sera utilisé.

bloc ECRITURE ETIQUETTES NOMMEES :
'ECRITURE ETIQUETTES NOMMEES'
'<mode>'
'<nom de référence>'

le mode doit être soit vide (") soit 'DUMP' (dans ce cas les étiquettes seront rangées sans conversion préalable. Le nom de référence identifiera l'ensemble des étiquettes nommées qui sera rangé et qui pourra par conséquent être utilisé pour des appels ultérieurs. Le nom de référence correspond à un nom de fichier relatif au répertoire "MODINT". Dans le nom de référence, le caractère "%" sera éliminé, le caractère "*" sera remplacé par le nom du fichier traité, et le caractère "#" par le nom du fichier sans l'extension. Si le nom du fichier n'existe pas le nom "MODINT/FORMATS.sygeft" sera utilisé.

bloc AGATE :
'AGATE'
'<nom de grammaire>' '<nom de dictionnaire>' '<nom de définition>'
Numéro de dimension concernée par le parcours.
'X' type de parcours.

X=1 parcours canonique avec appel avant et après le passage sur un point. X=0 parcours canonique avec appel sur les feuilles.

'X' visualisation de l'exécution.
'X' mode de visualisation.

bloc NOM FORMATS SORTIE :
'NOM FORMATS SORTIE'
'<nom du fichier>'

Ce nom de fichier sera utilisé si un fichier de formats internes est traité en écriture par le système. Le nom de référence correspond à un nom de fichier relatif au répertoire "MODINT". Dans le nom de référence, le caractère "%" sera éliminé, le caractère "*" sera remplacé par le nom du fichier traité, et le caractère "#" par le nom du fichier sans l'extension. Si le nom du fichier n'existe pas le nom "MODINT/FORMATS.sygeft" sera utilisé.

bloc NOM FORMATS ENTREE :
'NOM FORMATS ENTREE'
'<nom du fichier>'

Ce nom de fichier sera utilisé si un fichier de formats internes est traité en lecture par le système. Le nom de référence correspond à un nom de fichier relatif au répertoire "MODINT". Dans le nom de référence, le caractère "%" sera éliminé, le caractère "*" sera remplacé par le nom du fichier traité, et le caractère "#" par le nom du fichier sans l'extension. Si le nom du fichier n'existe pas le nom "MODINT/FORMATS.sygeft" sera utilisé.

bloc NOM MESSAGES SORTIE :
'NOM MESSAGES SORTIE'
'<nom du fichier>'

Ce nom de fichier sera utilisé si un fichier de chaines est traité en écriture par le système. Le nom de référence correspond à un nom de fichier relatif ou absolu. Dans le cas du nom relatif le fichier appartient au répertoire de lancement (ou à un sous-répertoire). Dans le nom de référence, le caractère '%' sera remplacé par le chemin du fichier d'entrée, le caractère "*" sera remplacé par le nom du fichier traité, et le caractère "#" par le nom du fichier sans l'extension. Si le nom du fichier n'existe pas le nom "MESSAGE.sygsrt" sera utilisé.

bloc NOM MESSAGES ENTREE :

'NOM MESSAGES ENTREE'

'<nom du fichier>'

Ce nom de fichier sera utilisé si un fichier de messages est traité en lecture par le système. Le nom de référence correspond à un nom de fichier relatif ou absolu. Dans le cas du nom relatif le fichier appartient au répertoire de lancement (ou à un sous-répertoire). Dans le nom de référence, le caractère '%' sera remplacé par le chemin du fichier d'entrée, le caractère "*" sera remplacé par le nom du fichier traité, et le caractère "#" par le nom du fichier sans l'extension. Si le nom du fichier n'existe pas le nom "MESSAGE.sygent" sera utilisé.

bloc NOM FICHIER RESULTAT :

'NOM FICHIER RESULTAT'

'<chaîne définition du fichier>'

Cette définition sera utilisée si un fichier est produit par le système AGATE différent du fichier 'RESUL'. La définition du fichier correspond à un nom de fichier relatif ou absolu. Dans le cas du nom relatif le fichier appartient au répertoire de lancement (ou à un sous-répertoire). Dans cette définition, le caractère '%' sera remplacé par le chemin du fichier d'entrée, le caractère '*' par le nom de ce fichier d'entrée, et le caractère "#" par le nom du fichier sans l'extension. La valeur par défaut est "%*.syg" et est utilisée dans le cas où ce bloc n'est pas défini.

Un fichier paramètre doit toujours débuter par un bloc "OPALE" où "LECTURE STRUCTURE".

La commande d'exécution (sous UNIX) est la suivante:

sygmart <Messages> [[<paramètres>] [-t] <texte> [-c | [-n | -p] [-L<nb>]]] [-i]

- Les paramètres doivent être donnés dans l'ordre.
- Si le nom du fichier paramètre est absent alors le fichier MODINT/PARAMSYGM doit exister et contenir les différents paramètres d'appel. Si seul le fichier texte est présent le paramètre "-t" est obligatoire.
- La forme d'appel "sygmart <fichier messages> " permet la définition d'un filtre:
  1. le fichier paramètre est bien sûr "MODINT/PARAMSYGM".
  2. le fichier d'entrée est le fichier standard d'entrée.
  3. le fichier de sortie est le fichier standard de sortie.

  L' exécution se termine par une fin de fichier.
- Le paramètre qui suit la définition des fichiers paramètre et texte d'entrée concerne la forme du listing et du résultat.
  1. Si le paramètre -c est présent les sorties s'effectuent dans le fichier standard de sortie.
  2. Si le paramètre -n est présent les sorties s'effectuent dans les fichiers SORTIE, LISTE et RESUL.
  3. Si le paramètre -p est présent toutes les sorties s'effectuent dans le fichier SORTIE.
  4. Sinon les sorties s'effectuent dans les fichiers SORTIE et <fichier texte>.syg ou un autre fichier si le bloc 'NOM FICHIER RESULTAT' a été défini.
  5. Dans le cas où un fichier est créé le paramètre -Ln indique le nombre de lignes par page.
- Le paramètre "-i" produit la liste des règles avec une marque (une étoile en fin de nom) indiquant celles qui ont été appliquées au cours de l'exécution dans le fichier standard d'erreur.

6.3 Les programmes de services

6.3.1 La génération du fichier des messages

Tous les programmes utilisent le m ême fichier pour identifier les mots clés ou les messages d'erreurs. Ce fichier est généré par le programme CRMSGER. La syntaxe est la suivante :

> CRMSGER <nom fichier résultat > <nom fichier caractéristique des caractères > <nom fichier sources messages internes> <nom fichier sources messages systèmes>

Les fichiers sources messages d'origine sont FICHMSG.h et FIMSGUNIX.h pour le sysème UNIX et le codage latin1 ou utf8, FIMSGUNC.h pour le codage interne utf16. Le fichier caractéristique des caractères est créé par le programme CRFICHCAR.

La génération du fichier caractéristique

Le programme CRFICHCAR génère un fichier contenant les caractéristiques des différents codages. La syntaxe d'appel est la suivante :

> CRFICHCAR <nom fichier résultat > [<largeur minimum en octets des caractères internes> <largeur minimum en octets des caractères systèmes> [< type d'encodage interne (F|V) > [< type d'encodage systeme (F|V)>]]]

- l'appel sans paramètre facultatif est équivalent l'appel CRFICHCAR <nom fichier > 1 1 F F
- l'absence d'un type d'encodage est équivalent à l'appel avec F

Trois fichiers sont originalement distribués correspondant au codage "Latin1" FICHCAR.h (paramètres 1 1 F F) "utf8" FICHCARUTF.h (paramètres 1 1 V F) et "utf16" FICHCARUNC.h (paramètres 2 1 V F), tous utilisant le codage ascii pour les messages systèmes. Avec ces éléments standard les fichiers des messages correspondants sont distribués :

- MSGR.latin1 : codage des caractères fixes de longueur 1 octet
- MSGR.utf8 : codage des caractères variables pour les données (grammaires, dictionnaires, textes, ...) de longueur minimale 1 octet et fixes de longueur 1 octet pour les messages systèmes (y compris les nom de fichiers).
- MSGR.utf16 : codage des caractères variables pour les données (grammaires, dictionnaires, textes, ...) de longueur minimale 2 octets et fixes de longueur 1 octet pour les messages systèmes (y compris les nom de fichiers).

6.3.2 Les programmes de compilation

Les programmes de services associés à la phase de compilation sont des programmes de tri des différents dictionnaires. Ces programmes ne fonctionnent que sur des fichiers syntaxiquement corrects. Trois programmes de tri permettent de trier les dictionnaires de formats, de chaînes ou d'étiquettes. La syntaxe est la suivante:

- tridcfm <nom fichier d'entrée> <nom fichier sortie> [latin1—utf8—unicode[<nom fichier ordre des caeactères>[nb]]]
- tridch <nom fichier d'entrée> <nom fichier sortie> [latin1—utf8—unicode[<nom fichier ordre des caeactères>[nb]]]
- tridctq <nom fichier d'entrée> <nom fichier sortie> [latin1—utf8—unicode[<nom fichier ordre des caeactères>[nb]]]

Le programme "tridcfm" trie les dictionnaires de formats, le programme "tridch" les dictionnaires de chaînes et le programme "tridctq" les dictionnaires d'étiquettes. Seules les parties terminales des dictionnaires de chaînes et d'etiquettes sont triées. La structure de ces dictionnaires demeure inchangée. Les noms des deux fichiers d'entrée et de sortie doivent être différents.
Les paramètres suivants permettent de paramétriser le tri. Le troisième définit le type de codage.
Le quatrième donne le nom du fichier définissant l'ordre à appliquer. Ce fichier comporte une ligne par

classe de caractères. La première ligne du fichier correspond à la classe de l'espace qui est un caractère de contrôle. Le rang d'apparition de chaque ligne correspond à l'ordre à définir. Chaque ligne comporte un ou plusieurs caract ères qui sont de même poids pour le tri initial. Ensuite deux éléments identiques suivant cet ordre sont classés suivant l'ordre des caract ères d'une même ligne. Ainsi, par exemple, si le fichier contient les lignes :
a
d
eéè
m
r
t
les mots "date, demis, démis, et démarrer" seront placés dans l'ordre :
date, démarrer,demis,démis.
La présence d'un espace à l'intérieur d'une ligne indique la définition d'un caract ère complémentaire. Cet espace est suivit de deux caractères : le caractère faisant l'objet d'un dédoublement et le caractère complémentaire qui sera ajouté pour le tri. Ainsi la ligne correspondant à la classe de 'a' pourrait être :
AÀÁÂÄ ÆEaàáâä æe
Le cinquième "nb", facultatif, définit le nombre maximum d'éléments à trier pour une zone terminale. Sa valeur par défaut est de 512000.

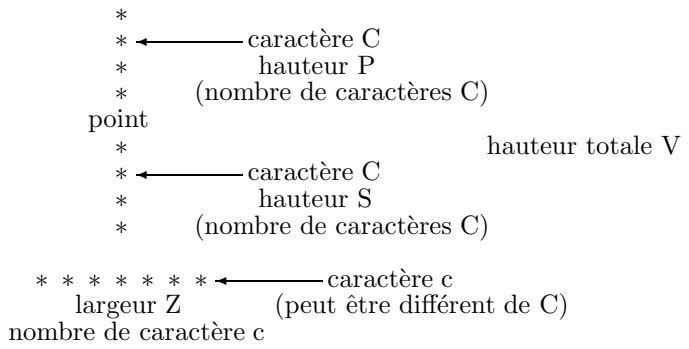
6.3.3 Les programmes d'exécution

Les programmes de services utilisent les éléments structurés définis par l'appel 'ECSTRUCT' de l'exécution. Ils ne font donc pas vraiment partie d'une application mais permettent une visualisation différente des résultats intermédiaires.

Le programme dessin

Ce programme permet de représenter les arborescences des éléments structurés. Chaque point de cette arborescence sera placé avec les références numéro de point, numéro d'étiquette et une valeur de variable optionnelle. Toutes les caractéristiques du dessin sont paramétrables et ce dessin peut être horizontal ou vertical.
Les différents paramètres sont les suivants:

dessin :



paramètres:

- Le premier paramètre est le nom du fichier des messages.
- Le deuxième paramètre est le nom du fichier contenant l'élément structuré. Ce paramètre est normalement de la forme MODINT/<nom de référence> si <nom de référence> est celui du bloc 'ECSTRUCT'.
- Les autres paramètres sont tous de la forme -x ou -xn avec comme valeur possible de x:

  H : nombre de caractères de la page en hauteur.

  L : nombre de caractères de la page en largeur.

Z : nombre de caractères en largeur réservés pour un point.

V : nombre de caractères en hauteur réservés pour un point.

P : nombre de caractères après le point.

S : nombre de caractères avant le point.

R : nombre de caractères après le point (dessin horizontal).

C : caractère vertical (-Ca où a est le caractère choisi).

c : caractère horizontal (-ca où a est le caractère choisi).

D : nom de la variable visualisée dans le dessin (-Dnom).

d : nom de la deuxième variable visualisée dans le dessin (-dnom), implique la présence de D.

I : visualisation horizontale.

N : pas de visualisation du contenu des étiquettes (dessin seul).

O : mode compressé.

G : La sortie n'est pas séparée en pages.

F : Le résultat se trouve dans le fichier SORTIE et non sur l'écran.

X : Le résultat correspond à une forme acceptable pour une visualisation X. Le dessin est défini par ses coordonnées.

p : La sortie s'effectue par sous-structure du niveau défini par ce paramètre (-p2 : niveau 2, la racine correspond au niveau 0 et les niveaux inférieurs à cette valeur ne sont pas visualisés).

Il existe certaines contraintes entre ces valeurs. Pour la visualisation verticale il faut nécessairement:

S + P + 3 = V

et pour la visualisation horizontale :

S + P + R + X = V (X valant 0, 1 ou 2 suivant la présence des paramètres D et d).

Si ces contraintes ne sont pas réalisées il y a une erreur de paramètres et le programme ne fournit aucun résultat.

Le programme resopa

Ce programme visualise le seul contenu des étiquettes des feuilles. Il repère ces feuilles par un contenu de variables. Dans ce cas deux feuilles de même niveau se suivent sans repère supplémentaire. Les paramètres sont similaires aux paramètres du programme dessin:

- Le premier paramètre
- Le deuxième paramètre est le nom du fichier contenant l'élément structuré. Ce paramètre est normalement de la forme MODINT/<nom de référence> si <nom de référence> est celui du bloc 'ECSTRUCT'.
- Les autres paramètres sont tous de la forme -x ou -xn avec comme valeur possibles de x:

  H : nombre de caractères de la page en hauteur.

  L : nombre de caractères de la page en largeur.

  Z : nombre de caractères en largeur réservés pour un point.

  D : nom de la variable dont la valeur sera utilisée pour l'identification de l'étiquette.

Le programme gentbvar

Ce programme permet la mise en oeuvre d'une communication inter-application par l'utilisation des formats d'entrée/sortie. Ce programme fournit sous forme de structures imbriquées la représentation interne des variables et de leurs valeurs. Le format sous sa forme binaire produit par la procédure d'entrée/sortie est composé de l'index de la définition en cours, d'une suite de chaînes de caractères de longueur fixe représentant le codage des variables différentes des variables de type chaîne, et des couples (longueur, chaîne) pour les variables de type chaîne. Le fichier produit par le programme "gentbvar" est sous le répertoire MODINT et porte le nom "TABLEVAR". C'est un fichier texte qui pour chaque définition produit l'index de cette définition, le nombre de variables globales de cette définition et deux blocs "MASQUE" et "CHAINE". Le bloc "MASQUE" définit la longueur de la représentation interne des variables globales et pour chacune d'elles l'ensemble des variables qui la composent avec leurs places dans la représentation interne (Origine et Longueur en nombre de bits). Pour chaque variable élémentaire l'ensemble des valeurs est énuméré sous la forme (Nom, Origine (de nature différente suivant le type de variable), valeur (en hexadécimal)). L'origine est soit en nombre de bit (variable potentielle), soit en nombre d'octets (variable exclusive) soit en nombre d'entiers de 4 octets (variable non exclusive). Lorsqu'une variable potentielle a le même ensemble de référence qu'une autre variable potentielle l'ensemble des valeurs n'est pas énuméré et le nom de cette variable de référence remplace l'ensemble des valeurs. Le nombre de masques et de chaînes produits pour un format d'une d'éfinition donnée est fixe et égal au nombre de variables globales de cette définition. Dans le cas ou une variable globale ne contient pas de variable de type chaîne une chaîne de longueur nulle est quand même produite. De même si une variable globale ne comprend qu'une variable de type chaîne un masque vide est produit. Un programme externe peut donc préparer une valeur d'étiquette et la transmettre par l'intermédiaire du fichier "MODINT/FORMATS.sygeft" au sous-système TELESI. Les fichiers "FORMATS.sygeft" et "FORMATS.sygsft" doivent appartenir au répertoire MODINT. Ils peuvent bien sûr être définis comme des "pipes nommés" pour constituer des applications compactes.

Le programme trdessin

Ce programme permet de traduire un fichier de détail d'exécution. Il utilise sensiblement les mêmes paramètres que le programme dessin. Le fichier source ne doit pas comporter de saut de page (produit avec le paramètre -L0). Les paramètres sont:

L : nombre de caractères de la page en largeur.

Z : nombre de caractères en largeur réservés pour un point.

V : nombre de caractères en hauteur réservés pour un point.

P : nombre de caractères après le point.

S : nombre de caractères avant le point.

R : nombre de caractères après le point (dessin horizontal).

C : caractère vertical (-Ca où a est le caractère choisi).

c : caractère horizontal (-ca où a est le caractère choisi).

I : visualisation horizontale.

N : pas de visualisation du contenu des étiquettes (dessin seul).

O : mode compressé.

F : Le résultat se trouve dans le fichier SORTIE et non sur l'écran.

X : Le résultat correspond à une forme acceptable pour une visualisation X.

x : Le résultat correspond à une forme acceptable pour une visualisation X. La non visualisation de la structure linéaire avec le dessin constitue la seule différence avec le paramètre 'X'.

p : La sortie s'effectue par sous-structure du niveau défini par ce paramètre (-p2 : niveau 2, la racine correspond au niveau 0 et les niveaux inférieurs à cette valeur ne sont pas visualisés).

f : Ce paramètre est immédiatement suivi du nom du fichier d'entrée. Si ce paramètre est absent le fichier d'entrée est le fichier standard.

6.3.4 Caractéristiques de mise en oeuvre

Les principales limites du système sont les suivantes:

entrée/sortie: Les entrées/sorties dans TELESI ne sont accessibles que sous le système UNIX. Les fichiers "FORMATS.sygeft" et "FORMATS.sygsft" doivent appartenir au répertoire "MODINT". Ces fichiers peuvent bien sûr être des fichiers "pipe nommés".

Variables: La place mémoire associée à une variable globale est fixe. Le nombre de variables à l'intérieur d'une variable globale est donc limité. Ce nombre dépend de la nature des variables qui la composent. Le nombre de bits associés à une variable globale est de 256. Les places occupées par les différents types de variables sont:

arithmétique : 16

arithmétique long : 32

flottant : 32

double : 64

exclusive : log2(n) + 1 si n est le nombre de valeurs.

non exclusive : n si n est le nombre de valeurs.

potentielle : 16 (sans précision) ou 24 (lorsque le nombre est défini).

référence : 32

La taille des variables de type chaîne est également limitée. Les tailles maximales sont les suivantes:

chaîne : 8192 caractères pour les variables et 1024 caractères pour les constantes.

variable externe : place occupée limitée à 8192 caractères.

Le nombre de variables globales est limité à 32765. Le temps d'exécution est pénalisé par un nombre de variables globales trop important. Il est raisonnable de ne pas dépasser 16 ou 32 variables globales.

Elément structuré: Le nombre total de points des éléments structurés est lui-même dépendant du nombre de variables globales. Ce nombre est de 2 milliards environ. Il doit être divisé par le nombre de variables globales (soit environ 68 millions pour 32 variables globales). Ce nombre constitue la capacité théorique. Il est bien sûr dépendant des mémoires disques disponibles. Il faut noter que les structures sont dupliquées dans les différents appels de grammaires TELESI. Donc la limite théorique d'un élément structuré manipulable correspond à environ 100 000 éléments pour 32 variables globales et un millier de grammaires TELESI environ.

Formats: Le nombre de formats dépend également du nombre de variables globales. Il est limité à 32 000 divisé par le nombre de variables globales.

Dictionnaires: Le nombre théorique d'éléments terminaux d'un dictionnaire de chaînes dépend également du nombre de variables globales. Il dépend bien sûr de la capacité des unités disques utilisés. Il est égal à environ 2 milliards divisés par le nombre de variables globales.

Le nombre théorique d'élements terminaux d'un dictionnaire d'étiquettes dépend de la variable utilisée pour l'adressage. Avec une variable d'adressage arithmétique ou potentielle longue ce nombre théorique est le même que celui défini pour les dictionnaires de chaînes.

Grammaire OPALE et AGATE: Le nombre de règles maximum est d'environ 32000.

Grammaire TELESI: Le nombre de points de l'ensemble des schémas d'une grammaire élémentaire est limité à 8192. Ce nombre correspond à la dimension de l'état du transducteur. Ainsi ce nombre peut être dépassé sans inconvénient dans le cas où plusieurs points de la même dimension des différents schémas sont incompatibles. Le compilateur ne peut bien sûr pas déceler cette incompabilité et fournit donc un message d'avertissement lorsque ce nombre est atteint.

Le nombre maximum de points distincts d'un schéma de reconnaissance est limité à 8192 pour

l'ensemble des dimensions et également à 8192 pour une dimension.

Le nombre total de points de l'ensemble des schémas d'une grammaire TELESI n'est pas limité pratiquement (supérieur à 10^9 si la mémoire le permet).

=-=-=-=-=-=-=-=-=-=-=-=-=

Index