UNIVERSITÉ DE NANCY I

U. E. R. DE SCIENCES MATHÉMATIQUES

INSTRUCTION D'AFFECTATION ET DÉFINITION D'UN MÉTALANGAGE DANS LE PROJET CIVA

THÈSE

pour obtenir le grade de

DOCTEUR - INGÉNIEUR

MATHÉMATIQUES APPLIQUÉES

par

BEN AM GHAR LAHOUCINE

Ingénieur E.N.S.E.M., Maître-ès-Sciences

Thèse soutenue le 30 Juin 1973 devant le jury :

MM.	PAIR C.	*Président*
	MARTIN J.	*Examinateur*
	DERNIAME J.-C.	*Examinateur*

UNIVERSITÉ DE NANCY I U. E. R. DE SCIENCES MATHÉMATIQUES

INSTRUCTION D'AFFECTATION ET DÉFINITION D'UN MÉTALANGAGE DANS LE PROJET CIVA

THÈSE

pour obtenir le grade de

DOCTEUR - INGÉNIEUR

MATHÉMATIQUES APPLIQUÉES

par

BEN AM GHAR LAHOUCINE

Ingénieur E.N.S.E.M., Maître-ès-Sciences

Thèse soutenue le 30 Juin 1973 devant le jury :

MM.	PAIR C.	*Président*
	MARTIN J.	*Examinateur*
	DERNIAME J.-C.	*Examinateur*

Ce travail a été réalisé sous la direction de Monsieur DERNIAME, que je tiens à remercier pour tous les conseils et les orientations fructueuses qu'il m'a prodigués.

Je tiens également à remercier, Monsieur le Professeur PAIR pour m'avoir fait l'honneur de présider le Jury.

Que Monsieur le Professeur J. MARTIN trouve ici, l'expression de ma plus grande reconnaissance pour le soutien qu'il m'a apporté tout au long de ce travail, de l'intérêt qu'il n'a cessé de me témoigner, et de l'honneur qu'il m'a fait en acceptant de faire partie du Jury.

Je remercie enfin, Mademoiselle LANG et Madame MARCHAND qui ont contribué à la réalisation pratique de cette thèse.

TABLE DES MATIERES

INTRODUCTION

Au moment de définir les différentes instructions du langage CIVA (affectation, entrées - sorties,.....) nous nous sommes trouvés dans l'impossibilité de couvrir tous les cas dûs à la diversité des objets du langage.

Dans l'exemple de l'instruction d'affectation, il serait ambitieux en définissant une (ou plusieurs instructions) de traiter l'ensemble des cas où chaque opérande peut avoir l'un des types suivants :

- type simple (entier, réel, booléen, décimal,.....)
- type file (avec taille variable, bornée, fixe)
- type structuré.

avec en plus, récursivement, les éléments des files et les feuilles des structures pouvant eu-mêmes être du type simple, file ou structuré sur un ou plusieurs niveaux de profondeur.

Certains langages de programmation comme Fortran IV, interdisent des opérandes de type file et structuré dans une affectation ;

D'autre comme COBOL et PL/1 les autorisent, mais figent les conventions de transfert pour l'utilisateur. Le fait de multiplier des instructions avec des conventions différentes (MOVE CORRESPONDING COBOL, affectation BYNAME PL/1) ne résoudrait pas entièrement le problème.

Nous pensons qu'en plus d'offrir à l'utilisateur des instructions couvrant les cas les plus usuels et les plus courants, il serait plus convenable de lui donner le moyen d'écrire et de définir lui-même les instructions appropriées à ses applications grâce aux extensions du langage CIVA (voir Partie II Méta-langage CIVA)

Ce qui a été dit à propos de l'affectation est valable pour les entrées-sorties d'éléments de type simple, file, structuré, mais aussi pour de fichiers, avec ou sans contrôle de validité,.....

(Voir Thèse de CHABRIER NANCY Juin 73)

La Ière partie de ce travail concerne la définition et le traitement

de l'instruction d'affectation dans CIVA (couvrant un nombre limité de cas).

La IIème partie concerne la définition du Méta-langage CIVA, ses spécifications et l'organisation du méta-processeur correspondant.

Ière PARTIE

AFFECTATION DANS CIVA

I - GENERALITES

1 - Objet de l'affectation

Elle permet d'attribuer une valeur à un objet identifié du langage.

Elle se décompose en deux parties

a - désignation d'un emplacement-mémoire, il suffit d'un identificateur d'emplacement

b - moyen d'accès à une valeur : qui peut être une dénotation de constante identifié, ou enfin le résultat d'un calcul.

2 - Syntaxe de l'affectation

⟨affectation⟩ :: = ⟨variable⟩ : = ⟨expression⟩

⟨expression⟩ :: = ⟨opérande⟩ /(⟨opérateur unaire⟩ ⟨expression⟩)/
(⟨expression⟩ ⟨opérateur binaire⟩ ⟨opérande⟩)

⟨opérande⟩ :: = ⟨variable⟩ / ⟨constante⟩ / ⟨ expression⟩ / ⟨appel de fonction⟩

⟨opérateur unaire⟩:: = +/-

⟨opérateur binaire⟩:: = +/-/* //**/

⟨variable⟩ :: ⟨variable simple⟩ / ⟨variable indicée⟩

3 - Rappels sur les objets traités dans CIVA

3. 1 - Fonctions d'accès

Nous distinguons les fonctions d'accès dans le langage et les fonctions d'accès en mémoire.

On peut atteindre une information dans le langage soit par accès direct (exemple emplacement d'une variable simple ou indicée x = F (4) ; y = F (I) ou F identifié un nom de file)

Soit par contiguité (exemple : POUR CHARUE A de F
FAIRE Y = Y + A fpc ;) où
l'adresse de l'emplacement d'un élément est calculé à partir de celle de l'élément précédent. C'est la fonction de succession dans les structures d'informations (cf. 27).

Pour la représentation interne des files et des structures et l'accès aux valeurs des éléments en mémoire, nous utiliserons d'autres fonctions telles que les chaînages, (voir implantation des objets en mémoire cf. 10).

3.2 - Notion de type attaché à un objet

Les valeurs que l'on peut manipuler dans un programme CIVA, peuvent être de diverses natures : les valeurs arithmétiques peuvent avoir des représentations binaires internes de différentes longueurs ce qui correspond pour les réels par exemple, à des précisions différentes ; de même le traitement d'un opérateur diffère selon que ses opérandes sont des entiers ou des réels.

Pour distinguer tous les objets ainsi manipulés dans le langage, on a associé à chacun d'entre eux un type ; au moyen de déclarations explicites.

Le type caractérise l'ensemble dans lequel un objet peut prendre ses valeurs ; (exemple : ensemble des entiers compris entre $-2^{+31} + 1$ et $2^{31} - 1$ ensemble des réels de la forme $\pm$ m.n E $\pm$ e ou le nombre de chiffres significatifs est au plus égal à 7 ou 16 etc...)

Types simples : entier, réel, booléen, décimal, caractère, complexe, réal précision, complexe double précision ;

Type file : file de taille fixe, file de taille bornée, file de taille variable ;

Type structuré.

Il faut y ajouter des types qualificatifs tels que procédure, module, classe ; ou de type introduit par le mot de base COMME.

3.3 - Variables simples et constantes

Elles suivent les mêmes règles que dans le langage FORTRAN CII 10070, pour ce qui est de leur construction syntaxique, limites, et notations.

3.4- Files dans CIVA

1 - Aspect sémantique d'une file

C'est une fonction d'accès qui à un p uple d'entiers associé un nom d'emplacement.

Exemple : file t [1: 10, 1: 5] entier ; pour $1 \leqslant i \leqslant 10$ et $1 \leqslant j \leqslant 5$

t [i, j] désigne un nom repérant un entier qui pourrait être éventuellement modifié par une affectation :
t [2, 3] = 4

On distinguera trois sortes de files en CIVA

- file de taille fixe
- file de taille bornée (ou maximum)
- file de taille variable

Dans les deux premiers cas la taille est connue à la traduction et la fonction d'accès est dite statique.

Dans le dernier, la taille de la file n'est connue qu'à l'exécution : la fonction d'accès est dite dynamique.

Pour l'implantation en mémoire (ou représentation interne) des files dans les différents cas voir travail de Mademoiselle LION (Thèse à paraître à NANCY) (cf. 13).

2 - Aspect syntaxique des files

<file ou sous file> :: = <nom de file ou d'ensemble> / <nom de file ou d'ensemble> ([<paire de bornes>][, [<paire de bornes>]]*)

<paire de bornes> : <bornes> : <borne>
<borne> : : = . / <expression arithmétique>

<nom de file ou d'ensemble> :: <identificateur>
<expression de file> :: = <désignation de file> / < expression de file >,

<désignation de file>

<désignation de file> :: = <file ou sous file> [<option d'ordre>] /
<expression arithmétique> / < suite de caractères> /
<expression booléenne>

/ (<expression de file>) [option d'ordre] .

<option d'ordre> :: = par ordre <sens>
<sens> :: = crois/décrois

Exemples :

F (15, 3: 17, 22) ; F1 (1: 15, 3: 17, 1: 22) ;

F1 (, 3: 12,) ≡ F1 (1: 15, 3: 12, 1: 22) ;

F1 (. : 12, 15: .,) ≡ F1 (1: 12, 15: 17, 1: 22).

Pour avoir des expressions de file on a muni l'ensemble des files d'une loi de composition interne: la concatenation notée "," ; exemple

Si A est une file d'entiers 1, 2, 3 et B la des 3 entiers 4, 5 et 6, alors l'expression A, B est la file 1, 2, 3,4, 5 6 ;

Une désignation de file pourra toujours être une file ou une sous-file, ou une expression arithmétique, ou une concatenation de ces dernières ;

Exemple :

F (1: 10), 'NOM-AJOUTE', F (11: 15) est une désignation de file ;

3.5 - Objets de type structuré

Une structure est une composition d'objets eux-mêmes de type simple, file ou structuré ; cette composition est généralement arborescente.

Pour accéder à une composante ou champ de la structure, on utilise un sélecteur ; une feuille est un champ dont la valeur est de type simple.

La sélection joue un rôle analogue à l'indexation pour les files. Un index peut être calculé alors que le sélecteur ne peut pas l'être.

Rappelons quelques distinctions entre un

objet de type file et un objet de type structuré.

- L'accès aux éléments simples dans une structure se fait au moyen d'un algorithme d'analyse d'arborescence, alors que pour les files, l'accès est réalisé par indexation et/ou indirection à partir de l'emplacement du premier élément.

- Le type des feuilles d'une structure peut être différent d'une feuille à une autre ; alors et chaque feuille est identifiée par un sélecteur. Pour les files, tous les éléments sont de même type et sont tous référencés par le nom de la file.

Exemple :

R structure (S entier, U réel, A file (5) car)

T structure (X entier, Y structure (V réel, W entier))

D'autres exemples sont donnés dans (13) avec des spécifications plus détaillées ainsi que l'ensemble des règles de syntaxe.

4 - UTILISATION ET CONVENTIONS DE L'AFFECTATION DANS CIVA

Nous énumérons ci-dessous l'ensemble des cas d'affectations ainsi que les conventions attachées à chaque cas.

Dans le chapitre suivant, nous détaillerons leur traitement.

Selon le type des opérandes, nous distinguerons deux cas.

4.1 - L'affectation élémentaire où les deux opérandes sont soit de type simple ou de type file d'éléments simples. (rappelons qu'un type simple est attaché à une dénotation de constante ou à une variable simple ou indicée et ce type peut être réel, entier, booléen, décimal, complexe réel, double précision, complexe double précision, caractère ; les files peuvent être de taille fixe, bornée ou variable.)

1 - Les deux opérandes sont de type simple. La valeur de l'opérande émetteur (à droite de l'opérateur d'affectation) est affectée à l'opérande récepteur (à gauche) , après une éventuelle conversion de type (voir chapitre suivant conversions autorisées dans CIVA).

2 - L'un au moins des deux opérandes est de type file.

2.1 - Récepteur de type simple - émetteur de type file
Cette affectation sera équivalente à l'affectation du premier élément de la file emettrice à l'élément récepteur.

2.2 - Récepteur de type file - émetteur de type simple
Cette affectation sera considérée comme l'affectation de l'élément simple émetteur au premier élément de la file réceptrice, si la file est de taille variable, sa nouvelle taille actuelle est égale à 1

2.3 - Emetteur et récepteur de type file

Quelque soit la nature des files (maximum, fixe ou variable), nous prendrons la convention d'affectater successivement aux éléments de la file réceptrice les valeurs des éléments de la file émettrice dans l'ordre de leur emplacement en mémoire.

On arrêtera le transfert dès que la file réceptrice sera "pleine" ou dès que tous les éléments de la file émettrice auront été "épuisés".

Si des éléments n'ont pas été affectés, leur valeurs resteront inchangées dans l'opération. Après une affectation les tailles actuelles des files variables ou bornées réceptrices prennent les valeurs des tailles des files émettrices.

4.2- L'affectation non élémentaire où l'un au moins des deux opérandes est de type structuré.

Nous nous limiterons au cas des structures dont toutes les feuilles sont simples (toutes les files éventuelles de cette structure ont été développées).

1 - L'émetteur seul est de type structuré

1.1 - Récepteur de type simple

Cette affectation sera considérée comme l'affectation de la valeur de la première feuille de la structure émettrice à l'élément récepteur.

1.2 - Récepteur de type file

Nous conviendrons d'affecter les valeurs des feuilles de la structure émettrice successivement aux éléments de la file réceptrice (aux conversions de compatibilité près)

L'ordre des feuilles étant celui de leur accès dans l'analyse de la structure (cf. Annexe - Analyse des Arborescences.) mêmes remarques que précédemment pour l'arrêt du transfert et la modification des tailles.

2 - Le récepteur seul est de type structuré

2.1 - Emetteur de type simple

La valeur de l'émetteur est affectée à la première feuille du récepteur (aux conversions près)

2.2 - Emetteur de type file

On affectera successivement les valeurs des élémehts de la file émettrice aux feuilles du récepteur dans l'ordre de leur accès respectif.

3 - Le récepteur et l'émetteur sont de type structuré

Cette affectation sera considérée comme la suite d'affectations élémentaires entre les feuilles des deux opérandes respectivement dans l'ordre de leur accès.

Remarque :

Comme il a été dit dans l'introduction, nous n'avons accepté qu'un nombre limité de cas d'affectation (correspndant aux restrictions essentielles suivantes : files ont toutes des éléments de type simples ; les structures ont toutes des feuilles de type simples).

Si l'utilisateur veut redéfinir les affectations précédentes (en modifiant les conventions de transfert) ou définir des affectations entre objets de type plus complexe (Ex : des files ayant des éléments eux-mêmes de type file ou structuré, des structure de files, il pourra le faire au moyen de méta-modules (voir IIème Partie))

II - TRADUCTION DE L'AFFECTATION CIVA

1 - Rappels sur les processus de compilation dans CIVA

A partir de textes sources écrits en méta-langage CIVA, le codifieur produit un texte intermédiaire (appelé chaîne codée) qui sert d'entrée au compilateur CIVA.

Le traitement d'une instruction d'affectation se fera en deux étapes :

Pendant l- codification, à la rencontre d'une instruction d'affectation, on produira les commandes de chaîne codée (voir annexe : format de la chaîne codée) correspondantes

Pendant la compilation de la chaîne codée, on générera le code objet associé.

Nous étuidierons d'abord le cas d'affectation élémentaire ou les deux opérandes sont de type simple ; et nous verrons comment tous les cas de l'affectation non élémentaire s'y ramèneront.

2 - Affectation élémentaire avec les deux opérandes de type simple

2.1 - Généralités et rappels

Soient op 1 et op2 respectivement, les opérandes de gauche et de droite de l'opérateur d'affectation "=" ; op2 peut être soit une dénotation de constante, de variable simple ou indicée dont le type est simple, ou plus généralement une expression arithmétique ou logique dont le résultat est de type simple.
op 1 est soit un identificateur de variable simple ou indicée dont le type est simple.

Dans CIVA, on traitera l'opérateur "=" comme un opérateur binaire quelconque (tel que + ,* , -, /,** ...) et on assimilera souvent une affectation de ce genre à une expression arithmétique généralisée.

La codification d'une expression arithmétique est donnée en annexe ; il s'agit essentiellement de la suite des descripteurs de chaque élément de l'expression dans l'ordre où ils apparaissent dans le texte source.

Une fois que le condifieur a analysé tout le texte source, le contrôle est passé au compilateur (analyseur et générateur).

Ce paragraphe n'a pour but que de préciser quelques problèmes posés par la traduction des expressions arithmétiques (cf. 18)

Pendant la génération du code objet correspondant à une expression arithmétique (supposée mise sous l'une des formes post fixée, pré-fixée ou infixée), on appliquera les règles suivantes :

- dans le cas d'un opérande indicé

Si les indices sont connus explicitement (constantes) Le compilateur détermine à ce moment l'adresse relative de cette opérande qu'il mettra dans sa pile de travail (cf. 18) auquel il adjoindra son type.

Si les indices sont des variables, le compilateur générera alors les ordres de calcul d'adresse de cet opérande (à l'exécution) et lui associera son type.

- Une fois qu'il a empilé tous les opérandes et tous les opérateurs et avant de générer les ordres de calcul de l'expression, le compilateur procède à des tests de compatibilité de type entre chaque opérateur et ses opérandes - (l'opérateur "=" inclus) et génère alors des ordres de conversions éventuels (qui se réduisent le plus souvent à des appels de sous-programmes fermés) ou imprime des messages d'erreur dans le cas d'incompatibilité.

La séquence à générer (conversion, rangement, libération éventuelles de mémoires temporaires) dépend des types des deux opérandes, (de leur longueur sous forme interne), mais aussi du fait que les valeurs à ranger se trouvent ou non dans des registres (cf. 11).

- Le compilateur peut procéder à d'éventuelles libérations de mémoires, de sauvegardes de registres.

- Le compilateur doit tester à la génération des instructions de rangement si une condition n'est pas attachée à la variable qui va être affectée., (aide à la mise au point dans CIVA cf. 18) et si oui générer la séquence des ordres correspondante à ce test ;

- Le traitement de l'opérateur binaire = diffère légè-

rement des autres : d'une part, on n'empile pas la valeur de l'opérande gauche mais son adresse ; d'autre part, une fois le traitement de cetopérateur terminé, les deux opérandes doivent être supprimés de la pile.

2.2 - Table des conversions autorisées dans CIVA

On a vu qu'avant de générer le code objet associé à la traduction d'un opérateur diadique, le compilateur doit effectuer des tests de compatibilité de type de ses opérandes. Il doit consulter la table suivante des conversions autorisées en CIVA : (Rq si deux opérandes ont le même type, aucun test n'est effectué).

	Entier	Réel	Booléen	Décimal	Caract	Complexe
Entier		OUI	OUI	OUI	OUI	OUI
Réel	OUI		OUI	OUI	OUI	OUI
Booléen	OUI	OUI		OUI	OUI	NON
Décimal	OUI	OUI	OUI		OUI	OUI
Caractère	OUI	OUI	OUI	OUI		OUI
Complexe	OUI	OUI	NON	NON	NON	

OUI : désigne conversion autorisée
NON désigne conversion non autorisée

2.3 - Priorité des types dans l'évaluation d'une expression arithmétique

Nous présentons ici deux possibilités de conversion de type dans une expression arithmétique.

1 - Cas du Fortran IV CII étendu (cf. 28)

On sépare l'opérateur = des autres opérateurs (+, -, , /, , etc....)
soit v = e une expression arithmétique généralisée
où v est un identificateur de variable simple ou indicée (de type simple) et e une expression arithmétique ne contenant plus d'opérateur =

si les opérandes composant e ne sont pas tous du même type que celui de V, elle est traduite séparément ; le résultat de e sera alors converti dans le type de V pour lui être affecté.

Dans le cas où e est elle-même mixte (contient plus d'un type), l'expression entière est évaluée dans le type de l'élément ayant la plus haute priorité ; cela signifie que les calculs relatifs à l'expression seront effectués dans ce mode, sauf en ce qui concerne les indices et les paramètres effectifs de fonction et exposant de puissance).

L'ordre de priorité est le suivant :

- complexe double précision 1 (plus élevé)
- complexe simple précision - Réel précision 2
- réel simple précision 3
- entier - 4

Dans une expression mixte (sans opérateur =) tous les éléments de plus basse priorité seront convertis dans le type de l'élément qui en a la plus haute, et ce avant d'être combinés aux autres éléments.

Les expressions apparaissant en indice, ou comme arguments de fonction sont indépendantes de l'expression où elles figurent ; elles sont évaluées dans leur type propre et n'affectent pas le type de l'expression qui les contient ni ne sont affectées par elle.

Les éléments de type complexe et réels double précision ont la même propriété, si une expression contient ces deux types, elle acquiert le type complexe double précision, c'est le seul cas où une expression peut avoir un type de priorité plus grande que celui de tous ses opérandes.

2 - Cas de CIVA

Soit v=e l'expression généralisée à traduire ;

Chaque fois qu'un opérateur doit être appliqué, il y a production d'une ou plusieurs instructions à 3 adresses ; lorsque l'opérateur a deux opérandes de type différent, on applique des règles de priorité sur les types des deux opérandes (et uniquement sur ceux-ci), pour générer des conversions éventuelles et ainsi de proche en proche, (voir module analyse des commandes expressions arithmétiques cf. 18)

Exemple :

A1, A2 réel, B1, B2 entier, C, D complexe
X entier

X = (A*B1) + (A2*C) - (B2/4.) + (A1*A2)

En Fortran IV CII A, B1, B2, A2, 4. seront tous convertis en complexe

puis l'expression de gauche est évaluée dans le mode complexe. Le résultat est alors converti en entier avant d'être affecté à X, en CIVA , on commencera par effectuer $\alpha 1$ = (A*B1) en convertissant B1 en réel

puis $\alpha 2$ = (A2*C) en convertissant A2 en complexe

puis $\alpha 3$ = B2/4. en convertissant B2 en réel.

puis effectuer $\alpha 4$ = A1 * A2 en réel

puis convertir $\alpha 1$ en complexe

puis effectuer $\alpha 5$ = $\alpha 1$ + $\alpha 2$ en complexe

puis convertir $\alpha 3$ en complexe, effectuer $\alpha 6$ = $\alpha 5$ + 23

convertir $\alpha 4$ en complexe, effectuer $\alpha 7$ = $\alpha 6$ + $\alpha 4$

convertir $\alpha 7$ en entier

affecter à X la valeur ainsi obtenue.

Remarque :

Dans les deux méthodes, pour l'opérateur =; le type du résultat est imposé par le type de l'opérande de gauche.

2.4 - Etude détaillée des conversions autorisées

1 - Entier ⟶ Réel

Elle est réalisée par le sous-programme fermé 9 ITOR, on génère l'appel à ce sous-programme chaque fois que l'on veut affecter une valeur entière a une variable de type réel (simple ou double longueur) ; on générerait l'instruction suivante :

BAL, 15 9 ITOR (si 15 est le registre de liaison).

Le paramètre effectif doit être placé dans le registre 9 ; le résultat est recueilli dans 8

BAL, 15 9 ITOD le résultat dans 8 et 9 ; le paramètre effectif étant placé dans 9.

2 - Entier ⟶ Booléen

Si l'entier est ≠ 0; le booléen prend la valeur VRAI. Sinon FAUX.

BAL, 15 9 I TOL

Le paramètre effectif est placé dans 8, le résultat 0 ou 1 est placé dans 8

3 - Entier ⟶ Décimal

Décimal condensé

On part de la représentation binaire interne d'un nombre entier pour aboutir à sa représentation décimale sous forme de chiffres codés un à un ; le nombre de chiffres est implicite et est égale à 31, (c'est la longueur maximum des décimaux que peuvent traiter les instructions de base de Métasymbol).

BAL, 15 9 ITODC

4 - Entier ⟶ Caractère

Si 0 ⩽ entier ⩽ 9 le caractère sera le code EBCDIC du chiffre F0 ⩽ code EBCDIC ⩽ F9
F0 étant un nombre héxadécimal sinon il y a impression d'un message d'erreur
On génére BAL 15, 9 ITOB

5 - Entier ⟶ Complexe

Il y a d'abord conversion de l'entier en réel, puis en complexe, on générera la séquence suivante :

BAL,15, 9ITOR

BAL,15, 9 ITOC ou BAL 15, 9 ITOK suivantqu'on veut passer en complexe simple ou double

6 - Réel ⟶ Entier

Le nombre représenté sous forme flottante sur un ou deux mots est converti en un nombre entier sur un mot, par des décalages sur sa mantisse et des réductions sur sa caractéristique pour la ramener à 0.

BAL, 15 9 RTOI

Paramètre effectif dans 8 ; résultat dans 9

BAL, 15 9 RTOL

8 - Réel ⟶ Décimal

Le réel est d'abord converti en entier, puis en décimal.

9 - Réel ⟶ Caractère

Le réel est d'abord converti en entier puis en caractère.

10 - Réel ⟶ Complexe

La partie réelle du nombre complexe est égale au nombre réel à convertir, sa partie imaginiaire est nulle.

9 RTOC ou 9 RTOK, ou 9 DTOC, ou 9 DTOK suivant que le réel est simple ou double et que le complexe est simple ou double.

11 - Booléen ⟶ Entier, réel, décimal

On fait associer à la valeur VRAI. La valeur 1 dans chacun des types récepteurs et à la valeur FAUX la valeur 0

12 - Booléen ⟶ Caractère

VRAI sera converti en le code EBCDIC du chiffre 1 ou de la lettre V ;

à FAUX sera converti en le code EBCDIC du chiffre 0 ou de la lettre F.

13 - Complexe ⟶ Entier, réel, booléen, décimal

On commencera par convertir le compte en réel et on est ramené à l'un des cas précédents. (BAL, 15 9 KTOR ou 9 CTOR).

3 - Affectation élémentaire mais l'un au moins des opérandes est de type file

La codification de cette instruction est immédiate et est analogue à celle du paragraphe 2 -
(Elle est donnée en annexe).

Quant à la traduction, on se ramène au cas des affectations entre objets de type simple (paragraphe 2)

3.1 - Récepteur de type simple -émetteur de type file

Si La file est du type file de caractères et le type du récepteur est autre que caractère, c'est la conversion en format libre d'entrée ; la file de caractère doit être nécessairement une dénotation de constante (Cf; 20) ; le traitement se réduira à la génération, à l'appel d'un sous-programme de conversion (équivalent du Décode Fortran).

Si La file est de type de carctère file et le type du récepteur est caractère ou la file est de type autre que file de caractères et le récepteur de type simple quelconque, On assimilera l'émetteur au premier élément de la file et on est ramené au cas d'affectation du paragraphe 2.

Exemple :

```
Entier X , F file (10) réel, G file (4) carct

X = F est équivalent à X = F (1)
si G contient les caractères 1, 2, 3, 4 alors
X = G équivaut à X = 1234
```

3.2 - Emetteur de type simple et récepteur de type file

Si la file est de type file de caractère, et l'émetteur de type autre que caractère, cette affectation réalise la conversion en sortie : partant de la représentation binaire

interne d'un élément pour obtenir la suite des chiffres et caractères (signe et point décimal éventuels) de sa représentation décimale ; selon des formats implicites ; la taille de la file doit être suffisante pour ne pas introduire d'erreur de troncature du résultat.

Dans les autres cas, le récepteur sera assimilé au premier élément de la file et on se ramènera au cas du paragraphe 2.

Il faut noter que si la file est variable ou maximum, sa nouvelle taille est égale à 1.

3.3 - Affectation élémentaire avec les deux opérandes de type file

Selon la nature des files, on distinguera trois cas :

- récepteur de type file de taille fixe d'éléments simples.

En aucun cas la taille de celle-ci ne sera modifiée dans une affectation ; si la taille de la file émettrice est plus grande que celle de la file réceptrice, elle sera tronquée pour obtenir le résultat ; si la taille de la file émettrice est plus petite seuls les éléments correspondants de la file réceptrice seront changés.

Exemple :

```
A file (25) car ; C file (10) car ;.....
A = A (1 : 15), C (ici la virgule désigne
l'opérateur de concatenation de file)
Remplace les derniers caractères de A par ceux
de C.

A = A (1 : 20), 'X', C remplace les cinq derniers
éléments de A par X suivi des 4 premiers éléments
de C.

A = A, C sans effet
```

A = C remplace les 10 premiers caractères de A par ceux de C.

- Récepteur de type file de taille maximum

La taille du résultat sera INF (taille actuelle de la file émettrice, la taille maximum de la taille réceptrice).

La nouvelle taille de la file réceptrice est celle du résultat.

Exemples :

A file (25) entier ; B file (max = 50) entier ; C réel

D (-7 : 1_) réel ; taille actuelle de B = 25

B = B (1 : 10), A : la taille actuelle de B devient 35 ; il n'y a pas de conversion, les valeurs des 25 éléments de A seront affectés aux éléments de B de rang 11 à 35.

B = B, C. La taille actuelle de B devient 36, la valeur de C est converti en entier avant d'être attribuée à B (36)

B = B (1 : 15) la taille actuelle de B diminue et devient égale à 15

B = B, D la taille actuelle de B passe à 38, la valeur de chaque élément de D est convertie en entier avant d'être affectée à l'élément correspondant de B

B = B, A la taille de l'émetteur a pour taille 63 elle sera donc tronquée ; la taille actuelle de B sera égale à sa taille maximum c. a. d 50.

Dans les deux cas : file de taille fixe ou maximum un indicateur sera positionné si l'affectation a nécessité une troncature de la file émettrice. La variable booléenne réservée à cet effet, IDEB FILE prendra la valeur VRAI, quand il y a débordement de la capacité de la file réceptrice, elle a pour valeur FAUX au début de l'exécution d'une chaîne de traitement et elle est remise à FAUX. Après chaque test, on peut l'attacher soit à toutes les files du programme, soit en créer une pour chaque file.

- La file réceptrice est de taille variable

Dans ce cas, c'est la file émettrice qui impose sa taille au résultat : la taille actuelle de la file réceptrice est modifiée et est égale à la taille du résultat.

Le cas des sous-files se ramène au cas des files après calcul de la taille et de l'adresse du premier élément à partir des indices des sous-files.

La codification d'une affectation entre deux files est immédiate : elle est une transcription codée de l'instruction du texte source.

A la génération du code objet, on connaît la nature de chaque file et le type de leurs éléments : si les tailles sont connues, le compilateur détermine leur minimum MIN et génère suivant que les types des éléments sont les mêmes. Une instruction de déplacement de MIN de caractères depuis l'adresse du premier élément de la file réceptrice, ou alors une boucle de MIN fois les instructions de chargement, appel de sous-programmes de conversion, rangement.

```
LI,5    MIN
LW, 3   EM, 5
BAL, 3  CONVERS
STW, 3  RE, 5
BDR, 5  %-3
```

(EM et RE désignent respectivement émetteur et récepteur).

Si les tailles ne sont pas connues (files de taille variables ou sous-files dont les bornes sont des variables), alors

Le compilateur doit générer les instructions de calcul du minimum des tailles ou/et du calcul d'adresse des premiers éléments des files avant de générer une séquence analogue à la précédente .

Remarques :

- Dans la séquence symbolique précédente, il est supposé que les éléments des deux files bien que de type différent ont la même longueur en forme interne (ils sont indexés par le même registre).
D'autre part, le transfert se fait élément par élément dans l'ordre de rangement des éléments en mémoire sans tenir compte de la façon dont sont déclarés les files (à 1 ou plusieurs dimension)

- Si les files ont des éléments de même type et de longueur interne, il serait plus efficace de générer une instruction de mouvements d'octets (MBS).

4 - Affectation non élémentaire

L'introduction des objets de type structuré conduit le plus généralement à des tables d'identificateurs trop grandes ; les analyses d'arborescences attachées à ces objets nécessitent l'existence de ces tables en mémoire ; on aurait pu garder ces tables jusqu'à la phase de génération.

La codification serait triviale et le compilateur analyserait lui-même les arborescences au moment de la génération du code objet. Il en résulterait les inconvénients suivants :

- une phase de codification serait injustifiée puisqu'elle consisterait en une simple transcription de texte (du moins en ce qui concerne les instructions d'affectation).
- L'espace mémoire alloué au compilateur se trouverait réduit du fait de la présence simultanée des tables de symboles et des descripteurs de localisation (cf. 10)
- La taille du compilateur se trouverait alourdie de sous-programmes de traitements supplémentaires (analyse d'arborescence,.....)

Dans CIVA, nous avons convenu de décomposer à la codification toutes les instructions d'affectation non élémentaires (au sens du paragraphe I - 4) en une ou plusieurs instructions d'affectation élémentaire vues précédemment, ce qui permettra de résoudre tous les problèmes d'adressage à la codification et de libérer l'espace alloué à une partie importante des tables des identificateurs (noms et localisation) qui sera utilisé à la compilation (piles de travail,....)

Donc chaque fois que le codifieur rencontrera une instruction d'affectation où l'un au moins des deux opérandes est de type structuré, il produit les commandes de chaîne codée (rubriques) correspondant à la suite des instructions élémentaires équivalentes.

Nous allons étudier les différents cas donnés au paragraphe I - 4.

1 - L'émetteur seul est de type structuré

1.1 - Récepteur de type simple

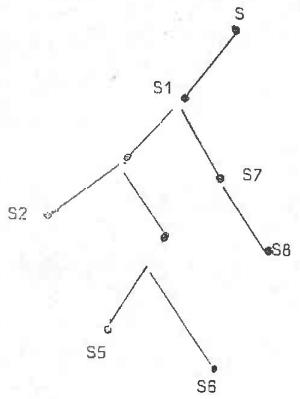
Le codifieur analysera l'arborescence de la structure émettrice et détermine le descripteur de la première feuille, il produit alors dans la chaîne codée la commande associée à l'affectation entre le récepteur et cette feuille

1.2 - Récepteur de type file

Le codifieur produira la suite d'affectations élémentaires correspondant au transfert de chacune des feuilles de la structure émettrice vers les éléments de la file, (l'ordre des feuilles étant celui de leur accès lors de l'analyse de l'arborescence associé à la structure cf. annexe 1).

Exemples :

Si F désigne une file de 4 caractères et S une structure dont l'arborescence est figurée ci-dessous.



alors F = G sera codifiée comme la suite d'affectations

F (1) = S3
F (2) = S5
F (3) = S6
F (4) = S8

Le nombre d'instructions produites est égal à MIN (nombre de feuille de S, taille de F), Si F est déclarée de taille fixe ; sinon, il est égal au nombre de feuilles de S ; dans ce cas, la nouvelle taille actuelle de F devient égale à ce nombre ; et on produira les commandes de cette dernière opération.

2 - Le récepteur seul est de type structuré

2.1 - Emetteur de type simple

Le codifieur produira la commande associée à l'affectation de la valeur de l'émetteur à la première feuille de la structure réceptrice.

2.2 - Emetteur de type file

Le codifieur produira les commandes de chaîne codée correspondant à la suite de transfert éléments de la file vers

les feuilles de la structure. Le nombre de commandes produites est égal dans tous les cas à MIN (nombre de feuilles de la structure, taille (actuelle) de la file).

Si nous reprenons la structure S et la file F de l'exemple précédent :

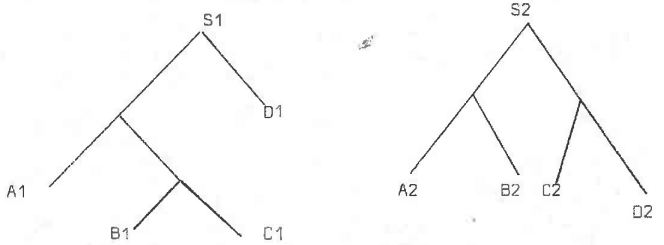
G = F sera codifiée comme la suite d'affectation

S3 = F(1) ; S5 = F(2) ; S6 = F(3) ; S8 = F(4)

3 - L'émetteur et le récepteur sont tous deux de type structuré

Le codifieur produira les commandes de chaîne codée correspondant à la suite des affectations respectives entre les feuilles des deux structures dans l'ordre de leur accès lors de leur analyse.

Exemple :



S1 = S2 sera codifiée comme la suite d'affectation
A1 = A2 ; B1 = B2 ; C1 = C2 ; D1 = D2

Le nombre de commandes produites est égal à MIN (nombre de feuilles de S_1, nombre de feuilles de S2).

ANNEXE 1 - ANALYSE DES ARBORESCENCES (STRUCTURES DANS CIVA)

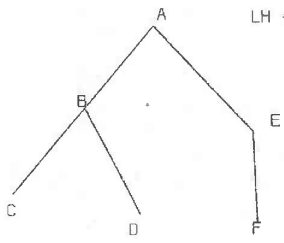
Dans CIVA, on associe à toute structure, une information arborescente, et nous nous limiterons aux structures à une seule racine.

L'analyse d'une arborescence est fonction de son implantation en mémoire. Elle consiste à passer d'un mot à un K - uple de valeurs ; le mot étant l'emplacement identifiant le nom de la racine de l'arborescence, et les valeurs étant celle des feuilles de l'arboresecence.

Parmi toutes les fonctions d'accès possible, nous avons choisi la représentation par un double chaînage (lien vertical et lien horizontal) (Voir Réf. 27).

Pratiquement, nous retiendrons dans la suite que le lien vertical d'un point d'une structure x est le premier sucesseur y de ce point dans la structure; et le lien horizontal d'un point x est le point y suivant dans la liste des sucesseurs où figure x.

Exemple :



LH = le lien horizontal de A = 0

LV (A) = B
LH (B) = E
LV (B) = C
LV (C) = D
LH (C) = D
LV (E) = F
LH (F) = LH (F) = LV (F) = LH (D) = 0

Tous les champs composant la structure figurent explicitement dans la table des identificateurs, avec des descripteurs de type, taille, localisation, lien vertical, lien horizontal dans la structure etc..... (Voir Réf. 10).

Chaque fois qu'on veut accéder à la valeur d'un champ d'une structure : on analyse l'arborescence de celle-ci ; jusqu'à rencontrer l'identificateur en question ; Nous utiliserons une pile dite attachée à la struture (on pourrait éventuellement s'en passer au prix de légères modifications du double chaînage ; en y ajoutant un pointeur supplémentaire : (Voir cours de PAIR structure d'information).

- au point de départ, la pile est vide, on y fait entrer la racine r ;
- après entrée d'un point x, on fait entrer r_1 (x) = LV (x) s'il existe sinon on fait sortir x de la pile ;
- après sortie d'un point x, tel que x, on fait entrer LH (x), s'il existe, sinon on fait une sortie de la pile ;
- après sortie d'un point x qui est la racine, on arrête l'analyse. Exemple (Voir Réf. 27).

ANNEXE 2 - FORMAT DE LA CHAINE CODEE ASSOCIEE A UNE AFFECTATION

La codification d'une instruction d'affectation consiste en la production d'un code suivi de la taille de l'expression et de la liste des descripteurs des opérandes et des opérations dans l'ordre où ils se trouvent à l'intérieur de l'expression ; les parenthèses, les virgules et les points virgules sont considéres comme des opérateurs.

Exemple 1 :

X = A + 7 - B*(C - D- - 40 ; la commande correspondante serait :

|03| 16 |descripteur de X| = |descripteur de A| + | descripteur de 7| - | descripteur de B|*| (| descripteur de C | - |descripteur de D |) |descripteur de 40 | ; |

Exemple 2 :

|03| 19 |descripteur de T |(|descripteur de I |) | = | descripteur de A | (|descripteur de I | , | descripteur de J |, |descripteur de 3 |) | + |descripteur de LOG | (| descripteur de 7 |) | ; |.....

Le descripteur d'une variable ou d'une constante est un mot contenant les renseignements sur le type, la classe ou le module auquelle elle appartient, sa valeur et son adresse. Les renseignements figurent dans le descripteur même dans le cas d'une constante ou d'une variable simple, sinon ; le descripteur sert de pointeur vers les tables contenant les renseignements cherchés.

PLAN IIème PARTIE

I - GENERALITES

1 - Définitions
2 - Concept de base des macro-assembleurs
3 - Concept de base de 5 méta-assembleurs

II - INTRODUCTION AU META-LANGAGE DE CIVA
Méta-processeur et Méta-traitement

III - META-LANGAGE DE CIVA

1 - Origine et but
2 - Méta-variables dans CIVA
3 - Méta-listes
4 - Méta-expressions
5 - Méta-instruction de base
5.1 - Généralités
5.2 - Méta-affectation
5.3 - Méta-instruction de saut
5.4 - Méta-instruction itération
5.5 - Méta-instruction conditionnelle
5.6 - Méta-instruction vide
5.7 - Méta-modules et méta-fonctions

IV - EXEMPLES D'ECRITURES DE META-MODULES

V - ORGANISATION GENERALE DU META- PROCESSEUR

IIème PARTIE

META-LANGAGE CIVA

I - GENERALITES

1 - Définitions

Soit L un langage de programmation ; une macro-instruction ou simplement, macro-dans le langage L est la désignation d'une suite d'instructions de L, par lesquelles cette désignation sera remplacée au cours de la traduction d'un texte source de L . La suite d'instructions s'appelle corps de la macro-instruction. La désignation s'appelle, référence à la macro.

Les corps des macros peuvent être : soit standard, soit écrits par l'utilisateur lui-même.

Elles sont considérées comme des sous-programmes ouverts dans le langage.

- Un langage L dans lequel, on peut ajouter des macros-instructions est appelé un macro-langage ;
- si le langage L est un langage assembleur, le macro-langage correspondant s'appelle macro-assembleur.
- Pour décrire et utiliser au mieux les macro-instructions de même que les instructions de L, on a introduit des méta-instructions.

Dans la plupart des cas, le concept de macro-instruction est lié au principe de la macro-substitution pure et simple, le concept d'une méta-instruction est lié au principe d'une macro-évaluation suivi éventuellement d'une macro-substitution.

La macro-évaluation ne donne lieu à aucune production de texte (objet ou source).

Si le langage est doté de telles instructions, il est dit méta-langage.(Réf. 1.2).

2 - Concept de base des macro-assembleurs

Initialement, le concept des macro-langages a été développé dans le cadre des langagesassembleurs, dont voici quelques propriétés.

Dans la plupart des applications les macro-instructions sont essentiellement utilisées comme extension du langage et permettent par ailleurs :

- réduction du codage des instructions
- souplesse et modularité des programmes
- facilité de mise au point et maintenance des programmes
- écriture standardisée des programmes
- une meilleure documentation des programmes ainsi que l'uniformisation des codes
- une meilleure vue d'ensemble des programmes sans rester au niveau des détails de l'instruction de base.

Principes du macro-traitement (cf. 6.7)

Le premier principe de base du macro-traitement est la production ou insertion d'une séquence de texte source dans une autre partie de texte source : c'est la génération de texte source.

Il faut noter que l'on ne tient compte ni de la forme, ni de la nature du texte à insérer. Le mécanisme de cette insertion se fait de la façon suivante :

dans le texte à analyser, on rencontre d'abord la définition de la macro-instruction (essentiellement son nom et l'ensemble du texte délimité qui la constitue : on dit son corps de définition) ; chaque fois que l'on rencontrera une référence à cette macro (ou appel de la macro) (essentiellement son nom placé à un endroit précis dans une ligne instruction du langage), Cette référence sera remplacée par le corps de défintion correspondant.

Une référence à un nom de macro peut apparaître n'importe où dans le programme (après sa définition).

Ainsi, la traduction du programme ne se fera qu'après remplacement de toutes les références de macros. (Ce remplacement peut éventuellement s'accompagner de traitements auxiliaires).

Exemple :

soit le texte source initial suivant où I1, I2.....In sont des instructions propres au langage ;

```
I1 ;
I2 ;
DEF MACRO 1     ligne de définition de la macro de nom
M1 ;            MACRO 1 suivie du corps la constituant ;
M2 ;
M3 ;
FIN MACRO 1
I3 ;
I4 ;
APPEL MACRO 1   ligne de référence à MACRO 1
I5 ;
APPEL MACRO 1   autre référence à MACRO 1
I6 ;
,
,
,
,
```

Le texte produit équivalent après macro-traitement est le suivant :

```
,
,
I1 ;
I2 ;
M1 ;
M2 ;
M3 ;
```

```
I3 ;
I4 ;
M1 ;
M2 ;
M3 ;
I6 ;
,
,
,
```

Le deuxième principe de base est la macro-substitution ou paramétrisation de texte (on dit aussi modification de texte).

En effet, il est rare qu'on recopie toujours intégralement la même séquence de texte à chaque appel d'une macro-instruction. Elle sert le plus souvent de modèle à partir duquel est construite la séquence de texte à produire à chaque appel.

La forme la plus simple et la plus fréquente utilisée, est l'adjonction d'arguments à chaque appel qui permet essentiellement de faire des modifications locales à l'intérieur du texte composant le corps de la macro.

Exemple : texte initial

```
I1 ;
I2 ;
DEF MACRO 2 ;
PARAMETRES : A, B, C ;
M1 ;
A ;
M2 ;
B ;
M3, B, C ;
FIN MACRO 2 ;
I4 ;
APPEL MACRO 2 ;
ARGUMENTS : E, F , G ;
```



```
I5 ;
I6 ;
,
,
,
```

Le texte résultant après le macro-traitement est le suivant :

```
I1 ;
I2 ;
I4 ;
M1 ;
E  ;
M2 ;
F  ;
M3, F, G ;
I5 ;
I6 ;
,
,
,
```

Les paramètres qui décrivent ainsi les procédures ne servent que pour indiquer la place des arguments effectifs lors de l'appel ; ils peuvent apparaître n'importe où dans le corps de la macro-instruction (et même n'importe où dans chaque zone d'une ligne instruction de ce corps).

Les arguments peuvent être n'importe quelle chaîne de caractères (exceptés quelques délimiteurs significatifs ex : le blanc ou la virgule,... point virgule....-

Rappelons les deux manières les plus souvent utilisées pour associer les paramètres effectifs de la ligne de référence aux paramètres formels de définition.

1 - Correspondance par la position respective des arguments dans la ligne d'appel et des paramètres dans la définition

2 - A chaque paramètre de la définition est attaché un mot clé et un numéro de position . Les mots clés servent à caractériser l'argument effectif et son paramètre formel correspondant.

Exemple :

```
DEF MACRO    ADD 2      &A, &B, &C
             CHARGER    &A
             AJOUTER    &B
             RANGER     &C
FIN MACRO

APPEL 1      ADD   A = 10, B = 25, C = X
APPEL 2      ADD   C = Z8, A = Y,  B = 49

Texte généré :  CHARGER 10
                AJOUTER 25
                RANGER  X
                CHARGER Y
                AJOUTER 49
                RANGER  Z8
```

Les mots clés sont les caractères du paramètre suivi d'un signe égal

- Avantages de cette dernière méthode

Elle permet d'écrire les arguments de la liste d'appel dans n'importe quel ordre, et peut fournir des programmes très clairs et documentés si les paramètres sont choisis et adaptés à la nature de la macro-instruction.

Elle permet de donner des valeurs par défaut aux paramètres qui n'ont pas de correspondant dans la liste d'appel ; ces valeurs par défaut sont directement associées aux paramètres dans la ligne de définition ; ceci étant particulièrement utile quand les paramètres prennent des valeurs particulières et sont rarement modifiées dans le reste du programme.

On verra qu'il existe une troisième façon d'associer les paramètres formels aux arguments effectifs par le moyen de la méta-fonction AF ; AF (I) désigne le ième argument effectif. (Voir plus loin méta-variables).

3 - Concept de base des méta-assembleurs

Lorsque les variations (ou modifications) que l'on désire introduire d'une ligne de référence à l'autre d'une même macro, sont un peu plus élaborées, la substitution ne suffit plus ; et on introduit alors d'autres éléments permettant de décrire les calculs à entreprendre pendant le traitement d'un appel de macro.

Parmi ces éléments, on distinguera les méta-variables puis les directives de méta-assemblage utilisant ces méta-variables, et enfin les macro-instructions appelées aussi procédures dans certains méta-assembleurs.

En plus, des principes que possèdent les macro-assembleurs, les méta-assembleurs permettent :

- la réalisation d'assemblage conditionnels : c. a. d générer une partie de texte ou une autre selon les résultats de conditions et de calculs qui ne figurent pas dans le texte produit.

- la réitération dans le texte produit d'une même séquence de texte source avec éventuellement des modifications locales.

- la réalisation de macro qui serait l'équivalent de sous-programmes fermés de type fonction ; (cas des procédures-fonction de Métasymbol cf. 1).

Nous verrons plus loin la terminologie équivalente dans le cas de Méta-langage évolués.

Pour les méta-assembleurs, nous garderons volontairement le nom de macro-instruction ou méta-procédure pour désigner une macro au sens du paragraphe 2.

Le terme méta-instruction (étant l'équivalent d'une directive d'un méta-assembleur, utilisant des méta-variables) étant utilisé dans un langage évolué.

II - INTRODUCTION DU META-LANGAGE DE CIVA

Les concepts du macro-traitement dans les méta-assembleurs peuvent aisément s'appliquer dans le cas de langages évolués.

En effet, le macro-traitement est essentiellement un traitement de texte, indépendant de la nature de ce texte.

Nous désignerons dans toute la suite par méta-module l'équivalent des macros ou des méta-procédures dans les macro-ou-méta-assembleurs.

- Méta-instruction l'équivalent des directives (ou pseudo-instructions) des méta-assembleurs qui permettent de décrire les méta-modules ;
- méta-langage, un langage pouvant accepter des méta-instructions (un appel de méta-module est une méta-instruction particulière).
- Méta-processeur, le programme chargé de traduire les textes sources comportant des méta-instructions.

Rappelons quelques propriétés des méta-variables.

Les méta-variables sont des variables dont la durée de vie est limitée au méta-traitement. Pour les méta-assembleurs, la phase du méta-traitement est le premier passage du processeur sur le texte (où il fait en même temps une étude lexicographique (constitution de la table des symboles). Le deuxième passage étant réservé pour satisfaire les références externes ou en avant et pour procéder à la génération du code objet.

Pour les méta-langages évolués, la phase de méta-traitement peut soit être indépendante de la phase de compilation proprement dite, soit intégrée dans celle-ci (voir méta-évaluation plus loin).

La portée d'une méta-variable s'étend à tout le reste du programme, excepté les corps des méta-modules qui déclarent cette variable locale. La portée d'une méta-variable déclarée dans un méta-module couvre le méta-module tout entier.

D'autres propriétés des méta-variables seront vues dans le paragraphe des méta-variables dans CIVA.

Pendant la traduction d'un programme écrit dans un méta-langage, on distingue trois phases distinctes :

a - la méta-évaluation : étape où une partie du texte source est considérée comme une donnée modifiable au moyen de méta-instructions. Le résultat de la méta-évaluation est une texte source dans lequel toutes les méta-instructions ont été traduites et qui servira de donnée au traducteur (processeur) du langage en question ;

b - évaluation syntaxique ou l'on réalise l'analyse syntaxique d'un programme source : celui-ci est transformé en un arbre syntaxique en utilisant les règles syntaxiques du langage ;

c - évaluation sémantique : qui permet la transformation d'un programme source dont l'évaluation syntaxique est correcte en un programme objet sémantiquement équivalent.

Nous donnons ci-après quelques précisions sur la méta-évaluation dans les langages évalués.

Comme dans le cas des langages assembleurs, la méta-évaluation (ou méta-traitement) est surtout basée sur un mécanisme de substitution et de paramétrisation ; elle sert surtout à étendre les possibilités du langage par l'introduction de nouvelles entités plus complexes n'appartenant pas au langage en question. Ce sont des méta-modules qui sont écrits par l'utilisateur et qui lui permettent des traitements modulaires et appropriés à son application.

Il existe deux formes distinctes de méta-traitement (Réf. 3) :

1 - La phase de méta-traitement est indépendante de la phase de traduction :elle précède alors obligatoirement celle-ci et consiste essentiellement en une génération ~~de texte~~ source. Le processeur de méta-traitement n'analyse que les méta-instructions qu'il rencontre en ignorant tout du reste du programme qu'il reproduit identique à lui-même. Il peut cependant supprimer les commentaires d'un texte si on le juge utile (Réf. 4), (Réf. 5).

Le méta-processeur effectue un passage sur le texte source, et ne gère que les méta-objets (méta-variables, méta-instructions, méta-modules) ignorant les autres éléments du texte qu'il reproduit tels quels.

Le texte résultant servira d'entrée au compilateur.

Dans cette forme de traitement, le méta-processeur a besoin de distinguer les méta-objets du reste du texte.

Avantage de cette forme :

Elle fournit un moyen à la fois simple, souple et efficace de définir des méta-instructions relativement complexes.

Inconvénient :

Elle nécessite un passage préliminaire d'analyse sur le texte initial et l'interaction possible avec le traducteur du langage est restreinte.

2 - La seconde forme (3) de méta-traitement consiste en un méta-processeur qui fait partie intégrante du processeur associé au langage (ici le processeur est un compilateur) ; c'est le cas en particulier de certains méta-assembleurs où l'on ne fait pas essentiellement de distinction entre méta-objets et objets du langage assembleur ; (Réf. 1)

Dans CIVA, on a opté pour cette deuxième solution

Avantages :

1 - Le méta-processeur de CIVA peut accéder à des informations dont dispose le compilateur CIVA lui-même. En particulier, il a accès à tous les renseignements concernant les objets du langage (on verra que ceci permettra d'écrire des méta-fonctions de base telles que type d'un identificateur, taille d'un identificateur, lien horizontal, etc....).

2 - Après le premier passage sur le texte source (c. a. d.) après la production de la chaîne codée), l'espace utilisé par une grande partie de la table des identificateurs (nom et localisations)

est restitué au compilateur pour la phase de génération : d'une part, les méta-objets ont leur durée de vie limitée au méta-traitement, d'autre part, les tables des objets langage ayant servi à produire le texte intermédiaire, ne sont plus totalement nécessaires. Toutes les substitutions auront été effectuées et les problèmes d'adressage relatif résolus ; seuls restent les descripteurs de type des objets CIVA et les tables des constantes.

Dans cette deuxième forme, le méta-traitement est incorporé dans le premier passage du compilateur, sur le texte source.

Dans CIVA, ce premier passage consiste en une étude lexicographique et la production d'un texte intermédiaire (ou chaîne codée) qui sera transformé lors d'un passage ultérieur en code objet proprement dit (phase de génération du code objet).

Lors de ce premier passage, on construit donc des tables des identificateurs des méta-objets en même temps que celles des identificateurs des objets CIVA. Pour cela, on analyse le texte origine du programme caractère par caractère depuis le début en délimitant les identificateurs et les instructions ou les méta-instructions.

Si on rencontre une instruction, elle est alors codifiée ;

Si on rencontre une méta-instruction, elle est immédiatement interprétée (ou exécutée). (Pour plus de détails, voir organisation du méta-processeur).

Pour reconnaître les méta-objets dans CIVA, on procéde de la façon suivante :

Chaque méta-variable doit avoir reçu une valeur au moyen d'une méta-instruction d'affectation ou d'itération avant d'être utilisée ; cette affectation constitue en même temps sa déclaration ; cependant, la méta-instruction % LØCAL permet aussi de déclarer les variables locales sans affectation.

(On aurait pu définir une méta-instruction de déclaration comme dans (4), mais nous pensons qu'une méta-variable n'a pas de type intrinsèque ; en un sens, il est dynamique puisqu'elle prend le type de la valeur

qui lui est affectée à chaque instant).

Chaque méta-instruction est précédée du symbole %.

Ceci a l'avantage de restreindre les recherches en table, et surtout de permettre à l'utilisateur une clarté meilleures des textes sources ("listings sources"). pour distinguer et reconnaître les méta-objets du reste ; (dans le but de mises au point diverses).

Chaque méta-module (ou méta-fonction) est reconnu par son identificateur au moment de sa définition qui précède nécessairement son appel (ou référence).

III - META-LANGAGE DE CIVA

- Origine et but du méta-langage CIVA.

On a vu que pour définir une affectation entre deux objets de type structuré ou de type files à plusieurs niveaux, nous étions amenés à choisir une convention bien déterminée au détriment de plusieurs autres :

Pour une affectation entre deux objets de type structuré, nous avons opté pour le transfert par le mot des feuilles ; (on aurait pu aussi ne transférer que les valeurs des champs ayant les mêmes noms dans les deux structures, ou les valeurs des champs de même niveau vertical ou horizontal,ou uniquement des objets du même type. Par ailleurs, on a volontairement écarté les objets de type plus complexe :

Exemple :

structures avec des champs eux-mêmes structurés ou de type files variables etc.....

Pour éviter ainsi de multiplier les instructions d'affectation correspondant chacune à un cas particulier, ou même d'élaborer une seule affectation dont le traitement prévoirait tous les cas possibles (ce qui serait ambitieux), on a été amené à étendre le langage CIVA de façon à permettre l'écriture de nouvelles instructions extérieures à CIVA, qui seront décomposées, à leur traduction, en des instructions du langage ; ces nouvelles instructions seront des appels à des méta-modules ou méta-fonctions.de CIVA

Ce qui vient d'être dit à propos des instructions d'affectations a été justifié aussi au niveau des instructions d-entrée/sortie du langage et de toute autre instruction de base du langage.

Le principe adopté peut se résumé ainsi :

On a défini dans le langage CIVA des instructions de base qui font un traitement déterminé, précis et on laisse à l'utilisateur la possibilité d'écrire un ensemble de méta-modules décrits à l'aide de ces instructions et qui lui permettent d'adapter au mieux les possibilités du langage à son application.

Il en résulte une plus grande souplesse de ses programmes. De plus, la conception des objets "méta" reste modulaire.

REMARQUE :

Dans CIVA, on appelle méta-module, les macro-instructions du langage. Pour décrire les calculs à entreprendre pendant la traduction d'un méta-module, on utilise des méta-instructions.

Dans la suite, nous allons voir en détail les objets du méta-langage, ainsi que les méta-instructions de base ; et nous donnerons des exemples pratiques de construction de méta-modules.

Comme le méta-processeur a accès aux autres objets du langage (du moins à leurs descripteurs, et non, à leurs valeurs), on adjoindra aux méta-instructions de base, des méta-fonctions de base telles que le type d'une variable, la taille d'un objet, son adresse relative dans une classe, les noms des éléments d'une structure, leur nombre, le nom des feuilles, etc....

2 - Méta-variables dans CIVA

- Durée de vie

- Une méta-variable est une variable dont la durée de vie est limitée à la phase de méta-traitement (dans CIVA, on l'appelle codification ou production de chaîne codée ou de texte intermédiaire).

- Syntaxe

- Elle suit les mêmes règles syntaxiques que les variables dans le langage. Elle peut être simple ou indicée; dans ce

dernier cas, c'est un élément d'une méta-liste (voir plus loin).

- Portée

Les méta-variables, contrairement aux variables ne sont pas déclarées explicitement, et leur portée s'étend à tout le texte du programme excepté les corps des méta-modules qui l'utilisent comme paramètre formal, ou qui les déclarent locales.

La portée des paramètres formels et des méta-variables définies dans un méta-module couvre celui-ci tout entier.

- Type et déclaration

Une méta-variable n'a pas de type proprement dit, mais nous dirons qu'elle possède le type de la valeur qui lui est affectée ; les valeurs que peuvent prendre des méta-variables seront exclusivement du type entier ou chaîne de caractères. La première apparition d'une méta-variable à gauche d'une méta-affectation ou comme variable contrôlée d'une méta-instruction d'itération (% FAIRE), constitue sa déclaration. Toute méta-variable doit avoir été déclarée avant son utilisation.

- Moment d'évaluation

Une méta-variable après avoir été déclarée peut apparaître n'importe où dans le reste du programme : aussi bien dans une méta-instruction, méta-module, que dans une instruction ou module.

- si elle apparaît dans une ligne d'instruction du langage, la valeur de la méta-variable de type entier est convertie en une chaîne de 8 caractères éventuellement précédée d'un signe ; les zéros non significatifs seront remplacés par des blancs, cette chaîne de caractères représentant des chiffres décimaux est alors insérée dans le texte produit et on continue l'analyse après la méta variable dans le texte original ; si elle est de type chaîne de caractères, sa valeur est recopiée dans le texte produit.
Le cas où elle apparaît dans une méta-instruction est traité au paragraphe "méta-expression".

- Reconnaissance d'une méta-variable

Lors de la déclaration d'une méta-variable, son nom est catalogué dans une table des méta-variables qui est tenue à jour tout au long de l'analyse du texte source.

L'apparition d'une méta-variable dans une méta-instruction, qui n'aura pas été déclarée auparavant , entraînera l'édition d'un message d'avertissement sur le texte produit et le méta-processeur ignorera cette instruction.

REMARQUE :

En aucun cas, une méta-variable et une variable ne peuvent être identifiées par le même nom.

3 - META-LISTES DANS CIVA (1)

Une liste est un ensemble ordonné d'éléments; chaque élément peut être identifié par sa position, le ième élément sera désigné par L (I) si L est le nom de la liste ; un élément lui-même peut être considéré comme une liste ; un élément d'une liste peut être une valeur numérique, une chaîne de caractères ou l'élément nul (c. a. d) (inexistants ou absent). UNe liste dont tous les éléments sont des valeurs numériques est dite liste de valeurs.

Une liste peut être linéaire ou non linéaire ; dans une liste linéaire, tous les éléments sont du premier degré (c. a. d., n'ayant pas d'élements eux-mêmes désignant une liste) ; une liste non linéaire possède au moins un élément identifiant une liste.

3.1 - Liste de valeurs

3.1.1. - liste de valeurs linéaire

Elle se compose d'un ou plusieurs éléments ayant une valeur spécifique non nulle de type entier.

REMARQUE IMPORTANTE :

Une méta-variable simple est identique à une liste de méta-variables à un élément, portant le même nom.

Exemple :

L = L (1)

Exemple de liste linéaire : L = 4, -3, 12, 7

3.2 - Syntaxe de référence et de définition

On définit une méta-liste par la méta-instruction d'affectation de la même manière que les méta-variablessimples. Cette définition sert en même temps de déclaration.

On écrira :

Ø L = 1, 3, -8, 0

(Le signe Ø sert à caractériser la méta-affectation)

Pour se référer à un élement d'une liste, on prend la notation indicée :

L (1) = 1 ; L (2) = 3 ; L (-3) = -8 ; L (4) = 0

Notons que L (n) = nul pour n > 4 ;

Ø A = 12, -8, +8 déclare A comme une méta-liste et la définit comme suit :

A (1) = 12 ; A (2) = -8 ; A (3) = 8 ;

A (p) = nul pour p > 3.

Un élément d'une liste peut être redéfini :

Exemple :

Ø A (4) = 10

Ø A (2) = 1

La nouvelle valeur de la liste A est

Ø A = 12, 1, +8, 10.

La valeur nul (différente de la valeur numérique 0) a dans une addition ou une soustraction le même effet qu'un zéro.

$ A = 15, 27, , , 9, 6 est identique à $ A = 15, 27, nul, nul, 9, 6

$ B = , , 4, 3 est identique à $ B = nul, nul, 4, 3

$ C = 8, 3, ≠ 8, 3, nul

Le nombre d'élements d'une liste dans CIVA est limité à 255

$ Q1 = 3

$ Q2 = 5

$ Q3 = Q1+Q2, Q-Q2-8, Q1 ≡ $Q3 = 8, -10,3

$ R = Q3 (1), 10, -1 ≡ $R = 8, 10, -1

$ R = Q3 ≡ $R = 8, 10, -1

$Q4 = Q3, 14, Q3, R (2) ≡ à $Q4 = 8, 10, 3, 14, 8, -10

$Q4 = Q4 (2) ≡ $Q4 = - 10

3.1.2 - listes de valeurs non linéaires

Exemple :

$ X = (4, 8) est non linéaire
si $ A = 1, 2, 4 (liste linéaire)
alors $ Y = A est aussi non linéaire

Les niveaux d'indices sont représentés par les groupements de parenthèses . Il arrive que les parenthèses soient redondantes.

Exemple :

Si $ A = ((2, 3, 4)), ((9, 8, 11)), 7 (6, (5, 4))
alors $ A (1) = (2, 3, 4)
$ A (2) = 9, 8, 11 sans parenthèses
$ A (3) = 7 $ A (4) = 6, (5, 4)
$ A (n) = nul pour n > 4
$ A (1, 1) = 2, 3, 4
$ A (1, 1, 2) = 3
$ A (2, 2) = 8
$ A (2, 5) = nul

$ A (4, 1) = 6
$ A (4, 1, 1) = 6
$ A (4, 2, 1) = 5
$ (4, 2, 2) = 4

Un ou plusieurs éléments d'une même liste non linéaire peuvent être redéfinis et réaffectés à d'autres valeurs, ou liste de valeurs

$ B = (5, 6), 72
$ A = 1,2,3, 14, B (3,1) $ A = 7, 17 car
$ B (3, 1) = nul
$ C = A, B = $ C = 7, 17, ((5, 6), 72)
$ C = A, (B) = $ C = 7, 17, (5,6), 72
$ C = (A), B = $ C = (7, 17), (5, 6), 72

3.2 - Listes d'éléments de type chaîne de caractères

$ A = 'B', 'KL', 'XYZ', '32' linéaire
si $ R = ('X', 'MOT', 'FIN'), (A(1), A(3)), A (2°
alors $ R = ('X', 'MOT', 'FIN'), ('B', 'XYZ'), 'KL'

On verra qu'une méta-liste peut avoir soit des éléments de type chaîne de caractères soit des éléments de type entier, ou des expressions d'éléments de ce type (voir expressions dans META-CIVA et conversions).

On adjoindra des méta-fonctions de base permettant de déterminer le type des valeurs des méta-variables si l'utilisateur désire éviter des méta-expressions mixtes (voir conversions - équivalent TCOR);

3.3 - Méta-fonctions associées aux listes

Ce sont des fonctions destinées à faciliter les traitements sur les listes pour l'utilisateur ; la plus fréquente est NUM (L) ou L est une liste ; elle détermine le nombre d'éléments non nuls (nul au sens absent) de la liste en paramètre ; l'utilisateur peut en écrire lui-même par exemple : NPOS (L) qui compte le nombre d'éléments positifs d'une liste. NNEG (L), NZR (L), NID (L) par exemple calculerai le nombre d'éléments identiques d'une liste ;

NIDX (LX) calculerait le nombre d'éléments d'une liste qui sont identiques à la valeur de la méta-variable X etc.....
SUP (L) et INF (L) calculeront la valeur du plus grand élément et du plus peit élément de la liste L.

4 - META-EXPRESSIONS DANS CIVA

4.1 - Opérandes

- Les méta-expressions sont des expressions dont les opérandes sont obligatoirement des méta-variables, ou des constantes entières ou des chaînes de caractères numériques ou des [illegible] ou des références à des méta-fonctions (de base ou écrites par l'utilisateur).

4.2 - Opérateurs et arithmétiques utilisés

Pour faciliter le travail du méta-processeur et rendre celui-ci moins complexe, nous nous limiterons dans ce méta-langage CIVA à une arithmétique entière sur des mots de 32 bits et des constantes entières d'au plus 8 chiffres décimaux, ainsi que des chaînes de caractères d'au plus 255 caractères.

Opérateurs arithmétiques + , -, *, / ; les seuls types de valeurs autorisés comme opérandes sont entiers; des chaînes numériques signées ou non seront d'abord converties [illegible] en entier (sous forme binaire) avant d'effectuer l'opération .

Exemple :

```
% I = 250
% J = I + 4500
% K = - 30*(I+J)
% K = 'AB' + 35 ⟶ entraînera une erreur
```

Opérateurs logiques de comparaison =, ≠, ≥, <, >, <

On distinguera deux types de comparaison

- comparaison entre des valeurs de type entier (sous forme interne binaire).
- comparaison entre des valeurs de type chaîne de caractères.

Les deux opérandes sont comparés de gauche à droite, caractère par caractère ; les chaînes de caractères sont cadrés à gauche et complétées par des blancs pour avoir le même nombre d'éléments.

Si un opérateur de comparaison fait intervenir deux valeurs de type différent (entier et chaîne de caractères) alors la chaîne de carctère est convertie en entier si elle représente une chaîne numérique (constante entière sous forme de chiffres signée ou non), sinon, il y a message d'erreur.

REMARQUE :

L'opérateur arithmétique = désignant une affectation a + un sens différent de l'opérateur = désignant une comparaison seul le contexte et l'instruction logique permet de les distinguer

```
% A = 'XY'
% B = 'XY'
% SI (A = B) alors..... : l'expression logique A = B vaut 1
% A = 'XY'
% B = 'ZT'
% A = % B . '4M' : cette instruction affecte à A la chaîne de caractères ZT4M
```

D'autre part, les autres opérateurs de comparaison ≥ , > , ≤ , < n'ont pas de sens pour des opérandes de type chaîne de caractères

Exemple :

```
% X = 'ABC'
% Z = '4FMCT'
```

La question X est-il plus grand que Z n'a pas de sens, on ne peut que répondre X est identique à Z ou X est différent de Z.

Ainsi les expressions logiques X > Z, X ≥ Z, X < Z, X ≤ Z et
X ≠ Z valent toutes 1
et X = Z vaudra 0
(On aurait pu s'interdire ce genre d'expressions, mais on pourrait aussi définir une relation d'ordre sur l'alphabet (A < B < C..... < Z)

Le résultat d'une opération de comparaison et plus généralement d'une expression booléenne sera la valeur 1 si elle est vraie sinon 0.

Les expressions booléennes seront plus fréquemment utilisées dans les méta-instructions (SI expression booléenne alors etc...)

- Opérateur de concatenation de chaînes

Il est noté "." et signifie que le résultat de l'opération A.B ou A et B sont des chaînes de caractères et une chaîne formée des chaînes A et B mises bout à bout.

Si A et B ne sont pas tous les deux de type chaîne de caractères, les valeurs de type entier seront converties en chaîne de caractères avant d'effectuer la concatenation (on aurait pu interdire la concatenation entre deux valeurs de type entier)

Exemple :

```
%L = 8
%I = 'AB'
%J = I . 'CDE'  ⟶  %J = 'ABCDE'
%K = 4 . L          %K = '4␣␣␣␣␣␣␣8'
```

La valeur interne de L est convertie en une chaîne de 8 caractères (le signe + et les zéros non significatifs sont remplacés par des blancs).

- Opérateurs logigues de réunion et d'intersection notés & et I

Les opérandes de ces opérandes sont nécessairement des expressions booléennes ; le résultat final d'une expression booléenne comprenant des opérateurs & et I sera la valeur 1 si elle est vraie,

0 sinon.

4.3 - Evaluation d'une méta-expression

Une méta-expression est évaluée au moment de sa rencontre par le méta-processeur du langage. LE traitement du résultat dépend de l'endroit où est apparue cette expression, mise à part l'opérateur d'exponientation non autorisé, les méta-expressions sont évaluées de la même façon que les expressions du langage.

- s'il s'agit d'une méta-affectation, la méta-variable de gauche prend la valeur de l'expression et on continue l'analyse (sous forme interne).

- s'il s'agit d'une instruction du langage, la valeur du résultat est convertie en une chaîne de caractères de longueur toujours égale à 8 caractères (chiffres décimaux avec signe) qui est insérée dans le texte produit et on continue l'analyse du texte original après la méta-expression.

- s'il s'agit d'une méta-instruction conditionnelle, ou d'itération, la valeur de la méta-expression est gardée dans une méta-variable temporaire créee par le méta-processeur (exemple le pas d'incrémentation, la valeur finale de la méta-instruction % FAIRE, la valeur d'une expression intervenant dans la méta-instruction % SI expression etc.....).

4.4 - Priorité des opérateurs

La liste suivante des opérateurs est donnée par priorité décroissante :

. , (* /), (+, -), (=, ≥, >, ≤, <), (&, I)

Les appels de méta-fonctions, ou calculs des éléments indicés et les groupements de parenthèses sont effectués les premiers.

5 - META-INSTRUCTIONS DE BASE DU LANGAGE CIVA

5.1 - Généralités

Dans le langage de CIVA, nous distinguerons les méta-instructions de base d'une part, et les méta-modules ou fonctions d'autre part. Toutes les méta-instructions sont précédées du symbole % ; et sont exécutées au moment de leur rencontre dans le texte source orignie. Elles comportent un symbole étiquette facultatif, suivi du corps de la méta-instruction et éventuellement d'un symbole de fin de méta-instruction (FAI E ; SI ;)
Comme dans tous les méta-langages, nous définirons :

- la méta-instruction d'affectation (notée =)
- la méta-instruction conditionnelle (SI)
- la méta-instruction de saut (ALLER A)
- la méta-instruction d'itération (ou de boucle (FAIRE)
- la méta-instruction vide
- [illegible] fonction de [illegible] texte [illegible] (ETIQUETTE)
- la méta-instruction de définition d'un méta-module
- la méta-instruction de définition d'une méta-fonction
- la méta-instruction d'appel d'un méta-module ou d'une méta-fonction
- les méta-instructions de fin de boucle, de fin de SI, de fin définition de méta-module ou de méta-fonction.

5.2 - Méta-affectation

Elle s'écrit % [ETIQUETTE :] partie gauche = partie droite.
Le symbole étiquette est facultatif, il sert à repérer l'instruction.

La partie gauche est soit une méta-variable simple ou indicée (élément d'une liste.)

La partie droite peut être soit une méta-expression, soit une ensemble de valeurs de type entier ou de type chaîne de caractères (le nombre d'éléments étant inférieur à 255),

ou de méta-expressions.

Exemple :

%I = 4	Constante entière sous forme de caractères
%J = 5, I+8, 4*I-5*3 , I/2	I liste de méta-expressions
%J = J+40-I	Méta-expression unique
%K = 'XY', 'A', 'BCD', 'EFGH'	liste de valeurs de type chaînes de caractères
%K = 'MMM' . K (3), 'C', 'RU'	
% ET1 : L = 1, 3, 5, 7	
% ET2 : L (4) = 8, 5	rédéfinit L comme la liste 1, 3, 5, 7, (8, 5)

Objet de la méta-affectation

Elle permet de déclarer une méta-variable en lui donnant une valeur de type entier ou chaîne de caractère.

Rappelons qu'une méta-variable ne possède pas de type intrinsèque comme les variables du langage qui elles, sont déclarées explicitement ; la méta-variable a le type de la valeur qui lui est affectée.

Exemple :

%I = 56	déclare I méta-variable de type entier
%I = 'AB'	redéfinit I méta-variable de type de chaîne de caractères

Il n'y a donc aucune conversion associée à une méta-affectation.

Lorsqu'une méta-variable a reçu une valeur, elle peut réapparaître n'importe où dans la suite du programme. deux cas peuvent se produire :

a - la méta-variable est attachée à une valeur de type entier

* Si elle est rencontrée dans une ligne instruction du

langage (même toute seule) et n'importe où dans cette ligne, elle est convertie en une chaîne de 8 caractères (signe, blancs, et chiffres significatifs) et elle est remplacée par cette chaîne dans le texte produit.

* Si elle est rencontrée dans une méta-expression, elle reste sous forme interne pour l'évaluation de la méta-expression

b - La méta-variable est attachée à une valeur de type chaîne de caractères

*Si elle est rencontrée dans une ligne instruction, elle est remplacée par la chaîne qu'elle désigne.

*Si elle est rencontrée dans une méta-expression (méta-expression de chaîne avec concatenation, ou comparaison, de deux chaînes dans la méta-instruction SI par exemple) elle entre dans l'évaluation de la méta-expression.

Exemple 1 - :

Soit le texte origine suivant :

```
%I = 10
%J = ' UTILISE C1, C2   '
%K = 'A ENTIER, B REEL', 'MOD'
%L = 'ULE', '48', '15', 'B = 9', ' M2 ;'
%M = K (2).L(1).L(5)
     K (1) ; J ;  M
%MA = I - 1000/I ;
A = I - 1000/I ;
B = L(2) + L(3) - L (4)
%MB + L(2)+L(3)-L(4)
C = L(2).L(3)
%MC = L(2).L(3)
D = MC+MB+MA
'
'
'
'
'
```

Dans le texte produit après le méta-processeur (s'il travaillait seul sans analyser les instructions du langage CIVA) serait :

```
A ENTIER, B REEL ;   UTILISE C1, C2 ;
MODULE M2 ;
A = 10 - 1000/10
B = 48 + 15 - 9
C = 48.15
D = 4815 + 54 - 90.
```

Exemple 2 - :

Remplacement sustématique de "constructions" d'un programme origine : cas des mots de base d'un langage FORTRAN.

```
% FAIRE = DØ ; % ALLER = GØ ; %A = TØ ; % SI = IF .
% LIRE = READ ; % ALØRS = THEN ; etc...
  LIRE (T) ;
  FAIRE I = 1A 49 ;
  SI T (I) < T (I+1) ALØRS ALLERA ETI ;
  etc....
```

Le texte équivalent et qui sera effectivement compilé, sera :

```
READ (T) ;
DØ I = 1 TØ 49 ;
IF T (I)<T(I+1) THEN GØ TØ ETI ;
etc...
```

Paramétrisation des programmes

```
%I = 1
UØ = 4
 X = I -3
 Z = 100/I
U = I**2 - LØG (I+U Ø)
```

On écrit les programmes avec une méta-variable qui sera partout remplacée par sa valeur unique du départ ; et s'il s'avère qu'il faut changer la valeur de cette variable, au lieu de la changer partout dans le programme on ne le change que dans une seule instruction, ce qui est valable aussi pour les noms des identificateurs.

Exemple :

%I = 'X'

5.3 - Méta-instruction de SAUT

Elle s'écrit % [ETIQUETTE 1 :] ALLERA ETIQUETTE 2 ;

Quand le méta-processeur rencontre cette instruction, il reprend son analyse du texte à partir de l'instruction ou méta-instruction repérée par ETIQUETTE 2 ; celle-ci pouvant être située avant ou après l'instruction ALLERA dans le texte origine.

Les symboles étiquettes n'ont d'existence que pendant le passage du condifieur (méta-processeur) et ils peuvent être n'importe quelle chaîne d'au plus 8 caractères (chiffres ou lettres) dont la première est obligatoirement une lettre.

5.4 - Méta-instruction d'itération

Elle s'écrit :

% [ETIQUETTE :] FAIRE V = e1, e2 [, e3] ␣ I1 ; I2 ; I3;.....IN ;

% [ETIQ 2 :] FIN FAIRE ;

V désigne une méta-variable qui est déclarée par la méta-instruction FAIRE.

e_1, e_2, e_3 sont des méta-expressions dont les résultats sont calculés une fois pour toutes à l'entrée de la méta-instruction et désignent respectivement la valeur initiale, la valeur finale et le pas d'incrémentation de la boucle.

I1, I2, I3 peuvent être des instructions du langage ou des méta-instructions.

Plusieurs, méta-instructions du type FAIRE peuvent être

imbriquées les unes dans les autres ; chaque boucle doit se terminer par un % FIN FAIRE.

Traitement :

e_1, e_2, e_3 sont d'abord évaluées

V reçoit la valeur du résultat de e_1 ;

On analyse le texte source formés des I1, I2,...IN ;

On calcule alors : V = V + e_3 ;

Si V $\geqslant$ e_2, le méta-processeur reprend son analyse après l'instruction % FIN FAIRE ;

sinon, on reprend l'analyse des instructions I1 jusqu'à IN, et ainsi de suite.....

si e_3 est omis, il est pris par défaut égal à 1

(on pourra envisager dans une version ultérieure de CIVA, une méta-instruction du type REPEAT de FORTRAN où les expressions e_1, e_2, e_3 sont recalculées chaque fois avant de rentrer dans la boucle).

Exemple 1 :

```
%L = 'A1=', 'A2=', 'A3=', 'A4=', 'A5=10'
%K = 3
%I = 'B = 40 ; C = 5 ;'
% FAIRE J = 1 ; K+1, K-2
I␣ X=J+1 ; Y=X-K  J ; L(J)␣J ; % FIN FAIRE ; L(5) ; etc...
```

Ce texte sera traité comme :

```
B = 40 ; C = 5 ; X = 1+1 ; Y = X - 3*1 ; A1 = ␣1 ;
B = 40 ; C = 5 ; X = 2+1 ; Y = X - 3*2 ; A2 = ␣2 ;
B = 40 ; C = 5 ; X = 3+1 ; Y = X - 3*3 ; A3 = ␣3 ;
B = 40 ; C = 5 ; X = 4+1 ; Y = X - 3*4 ; A4 = ␣4 ;
A5 = 10 ; etc....
```

Exemple 2 :

```
% FAIRE  I= 1, 10
Z (I) = X (I) - Y (I)*Z (I) ;
% FIN FAIRE ; A = 3 ; etc...
```

Texte équivalent

```
Z (1) = X (1) - Y (1) * Z (1)
Z (2) = X (2) - Y (2) *Z (2)
 ⋮
Z (10) = X (10) - Y (10) *Z (10)
A = 3 ; etc...
```

5.5 - Méta-instruction conditionnelle

Elle s'écrit :

% [ETIQ] : SI méta-expression % ALØRS { groupe 1 d'instructions ou de méta-instructions ;

[% SINON { groupe 2 d'instructions ou de macro-instructions ;]

% [ETIQ 1 :] FIN SI ;

La méta-expression peut être arithmétique ou booléenne ; le résultat d'une méta-expression booléenne est égal à la valeur 1 si elle est vraie, 0 sinon ;

Le méta-processeur évalue tout d'abord la méta-expression qui se trouve derrière % SI ; si son résultat a une valeur différente de 0, (positive ou négative), il y a analyse du groupe 1 d'instructions ou de méta-instructions qui suit % ALØRS, puis reprise de l'analyse du texte qui se situe derrière % FIN SI.

Si le résultat de la méta-expression est égal à 0, il y a analyse du groupe 2 d'instructions qui sont derrière le mot de base % SINØN suivi de celle du texte après % FIN SI ;

L'ensemble % SINØN groupe 2 instructions ou de méta-instruction peut être omis.

Plusieurs méta-instructions SI, peuvent être imbriquées ; chacune d'entre elles soit se terminer par un % FIN SI ;

Exemples 1 - :

```
%J = 5
%I = 1
% E1 : SI (I = J) & (I< 8)  % ALØRS C (I, J) = I + J ;
% SINØN C (I, J) = B (I, J) ; %I = I + 1 ; % ALLERA E1 ;
% FIN SI ; X = 5 ; Y = C (I, J-1) ; etc...
```

Texte équivalent

```
C (1, 5) = B (1, 5) ; C (2, 5) = B (2, 5) ; C (3, 5) = B (3, 5) ;
C (4, 5) = B (4, 5) ; C (5, 5) = 5 + 5 ; X = 5 ; Y = C (5, 5-1) ;
etc....
% L NØM = 'JEAN', 'PAUL', 'JACQUES', 'RENE', 'MARC'
% FAIRE I = 1, 6
% SI (J = L NØM (I)) % ALØRS X% = T (I) ; % SINØN
% A = A + 1 ; % FIN SI ; % FIN FAIRE ; Y = I etc...
```

Rappel N

Texte équivalent

```
X = T (4) ; Y = 6 ; etc...
```

5.6 - Méta-Instruction Vide

Elle s'écrit % ETIQUETTE : ;

Elle sert à repérer une ligne du texte source (instruction ou méta-instruction) et ne produit aucune action.

Texte origine

```
C = 4 ;
Ø X = 5
Ø ALLERA ET1 ;

Ø ET1 ; D (X, X) = C (I, J) ;
ØK = 17 ;
Z = 5 - P (X)
U = 3 + K ;
Etc...
```

Texte produit

```
C = 4 ; D (5, 5) = C (I, J) ; Z = 5 - P (5) ;
U = 3 + 17 ;
```


5.7. - META-MODULES ET META-FONCTIONS DANS CIVA

5.7.1. - Généralités

Munis des méta-instructions de base définies précédemment, l'utilisateur a la possibilité d'écrire des sous-programmes ouverts qu'il peut définir une fois pour toutes dans son texte origine, et appeler une ou plusieurs fois dans la suite sans réécrire l'ensemble des instructions qui les composent.

Ce sont des méta-modules ou méta-fonctions qui dont la durée de vie est limitée au méta-traitement. Ils sont équivalent de MACRO dans les langages d'assembleur.

5.7.2 - Méta-instruction de définition d'un méta-module

```
Ø DEF - MØD    NØM (Arg 1, Arg 2,.....) ;
               I1 ;
               I2 ;
               I3 ;
                '
                '
                '
Ø FIN  MØD     ;
```

NØM désigne une suite d'au plus 8 caractères identifiant le nom du module.

Arg 1, Arg 2,..... désigne la liste éventuelle des paramètres du module ; I1, I2,.....IN sont soit des instructions ou des méta-instructions.

de base ;

La définition d'un module doit se terminer par Ø FIN MØD ;

5.7.3 - Méta-Instruction de définition d'une méta-fonction

Ø DEF-FØNC NØM (Arg1, ARg 2,.....)
I1 ;
I2 ;
I3 ;
'
'
'

Ø FIN FØNC

NØM est l'identificateur du nom de la fonction

Arg 1, Arg 2,..... sont des paramètres formels (leur nombre est ⩾ 1)

I1, I2, I3,..... sont obligatoirement des méta-instructions dont l'une au moins consiste à affecter une valeur à l'identificateur NØM.

5.7.4. - Référence à un méta-module

Ø NØM (P1, P2,.....) ;

NØM est l'identificateur du module à appeler ;

P 1, P2,..... sont les paramètres effectifs :

ils peuvent être des méta-expressions, des objets du langage ou des chaînes de caractères quelconques.

Une ligne d'appel d'un méta-module forme à elle seule une ligne instruction.

5.7.5. - Référence à une méta-fonction

L'appel d'une méta-fonction se fait par la référence à son nom précédé de Ø et suivi d'au moins un paramètre effectif entre parenthèse ; Ø NØM (P1, P2,.....)

L'appel peut apparaître dans une méta-instruction (=, FAIRE, SI) dans une ligne d'instruction du langage CIVA, ou comme argument effectif d'un méta-module ou d'une autre méta-fonction.

Pour les paramètres effectifs P1, P2....., mêmes remarques que pour les méta-modules.

5.7.6. - Remarques sur les appels de méta-modules et des méta-fonctions

Les corps de définitions des méta-modules (ou méta-fonctions) ne sont pas conservés au moment de leur apparition.

A la rencontre d'un appel de méta-module (ou de méta-fonctions)

- On évalue les paramètres effectifs
- On analyse le texte correspondant instruction par instruction avec substitution éventuelle des paramètres effectifs évalués aux paramètres formels.

Pour un méta-module, on reprend l'analyse à partir du caractère suivant l'appel.

Pour une méta-fonction, on insère sa valeur à l'endroit de l'appel avant de reprendre l'analyse à partir du caractère suivant.

Les méta-modules de CIVA ne comportent que des instructions ou méta-instructions CIVA.

Les méta-fonctions ne comportent que des méta-instructions dont l'une au moins consiste à donner une valeur de type entier ou chaîne de caractère à l'identificateur de la fonction.

On ne peut sortir d'un méta-module (ou d'une méta-fonction que par la méta-instruction Ø FIN MØD (ou S/ FIN FØNC).

Ceci évitera de définir une méta-instruction de retour équivalente au RETURN FORTRAN qui serait exécutée pendant le méta-traitement.

EXEMPLES :

Soit le programme CIVA suivant :

A file (8) entier ; B file (6) entier ; C file (4) caract ;

```
X entier ; Y entier ; Z entier ;
X = 5 ;
Y = 8 ;
% DEF-MØD   INITAB (T, VAL, NEL)
% FAIRE I = 1, NEL ;
T (I) = VAL ;
% FIN FAIRE ;
% FIN-MØD ;
Z = X + Y
% INITAB (A, 0, 8) ;
% INITAB (B, 1, 6) ;
%J = '␣' ;
% INITAB (C, J, 4) ;
    '
    '
    '
  etc
```

Ce texte serait l'aquivalent de :

```
A file (8) entier ;..... ; Z entier ;
X = 5 ;
Y = 8 ;
Z = X + Y ;
A (1) = 0 ;
A (2) = 0 ;
A (3) = 0 ;
   '
   '
   '
A (8) = 0 ;
B (1) = 1 ;
B (2) = 1 ;
   '
   '
B (6) = 1 ;
C (1) = '␣' ;
C (2) = '␣' ;
C (3) = '␣' ;
C (4) = '␣' ;
  etc.....
```


5.7.7. - Méta-fonction de base dans CIVA

1 - Méta-fonctions de base jointes au méta-processeur ou disponibles dans des classes (voir instruction UTILISE CIVA).

Elles seront écrites directement en langage d'assembleur et permettent à l'utilisateur d'avoir accès à tous les renseignements constitués par le codifieur jusqu'au moment de l'appel de la méta-fonction ; (tables des identificateurs, tables de descripteurs, type , taille, lien vertical, lien horizontal, adresse de localisation relative dans la zone statique d'un objet du langage (élément simple, file structure,....).

2 - Les autres méta-fonctions que l'"utilisateur" peut définir à son gré dans ses programmes, en se servant éventuellement des précédentes et qui assurent une plus grande modularité, et une clarté à ses programmes qui seront écrites en CIVA.

Nous donnons ici la liste exhaustive des méta-fonctions de base de CIVA et leur description détaillée. Pour les informations concernant les structures, la description et la terminologie est donnée dans l'annexe 1ère partie).

TYPE (X) désigne le code du type de la variable X du langage. Le résultat est de type entier ;

TAILLE (Y) désigne le nombre d'éléments simples composant l'objet identifié par Y, il n'a de sens que pour les objets déclarés de type FILE fixe et désigne le nombre de ses éléments. Le résultat est une valeur de type entier.

Si Y n'est pas un objet de type file fixe, TAILLE (Y) = 0

LH (A,Z)désigne le nom de l'identificateur du champ lien horizontal du champ identifié par Z dans la structure identifiée par A.

A et Z peuvent être des méta-variables auxquelles ont été affectées des valeurs de type chaîne de caractères ou des chaînes de caractères.

LV (A, Z) désigne le nom du lien vertical de Z dans la structure A ;

ADR (X) désigne l'adresse relative de l'objet X dans la zone statique.

X peut être une constante (respectivement une variable) déjà répertoriées dans la table des constantes (respectivement tables des identificateurs)

RANG CHAMP (A, X) est associé au rang (valeur d'une méta-variable) qu'occupe le champ désigné par X dans la structure A, dans l'analyse de l'arborescence de A par liens veritcaux et horizontaux.

NØM FEUIL (A, I) est associé au nom de l'identificateur de la 1ère feuille rencontrée dans l'analyse de l'arborescence de A.

NUM (L) : désigne le nombre d'éléments présents dans la liste L

NB REFEUIL (A) : désigne le nombre de feuilles de la structure A

NØM CHAM (A, I) : désigne le nom du Ie champ de la structure A pendant l'analyse de son arborescence

RANG CHAMP (A, X) : désigne le numéro d'ordre du champ de nom X dans la structure A

NØM FEUIL (A, J) : désigne le nom Je feuille de la structure A pendant l'analyse de son arborescence

MAX (A, B) ; MIN (A, B) : désignent le maximum et le minimum des deux méta-variables A et B

A F (I) : cette méta-fonction d'accès dans la file des paramètres effectifs a pour valeur du ième paramètre effectif d'un méta-module ou d'une (méta-fonction) <u>après évaluation</u>.

Elle est utile quand le nombre de paramètres effectifs varie d'un appel d'un même méta-module à un autre. Elle se prête aussi très facilement aux calculs itératifs sur la liste des valeurs des paramètres effectifs.

BORNE INF (X, I) désigne respectivement la valeur de la borne inférieure

BORNE SUP (X, I) désigne respectivement la valeur de la borne supérieure

du domaine de variation du ième indice de la file X

si i est absent, il est égal par défaut à 1 ;
si la file a été déclarée sans bornes explicites, la borne inférieure est égale à 1 et la borne supérieure égale à la taille de la file dans ce dernier cas Ø BØRNE SUP n'a de sens que si la taille est connue à la compilation (c. a. d. uniquement dans le cas des files de type fixe).

<u>Exemple</u> :

F File (5) ; F1 <u>file</u> (8, 3) ; F2 <u>file</u> (5, 4, 2)

F3 <u>file</u> (-3:7), F4 <u>file</u> (3:5, -2:3)

Ø BØRNE INF (F) = 1	Ø BØRNE SUP (F) = 5
(F1, 1) = 1	(F, 1) = 8
(F1, 2) = 1	(F, 2) = 3
(F2, 3) = 1	(F2, 3) = 2
(F3) = -3	(F3) = 7
(F4, 1) = 3	(F4, 1) = 5
(F4, 2) = -2	(F4, 2) = 3

Ø NØM PRED (A, X) désigne le nom du prédecesseur du champ de nom X dans la structure A.

NB PREDEC (A, Y) désigne le nombre de prédecesseurs de Y, la racine A incluse dans l'analyse de l'arborescence de la structure A.

NBRE CHAM (A) détermine le nombre total de champs (élémentaires ou non) de la structure A, la racine A

RG SYMBØL (A, X) calcule le rang de l'élément de la liste A qui identique à la valeur associée de X

A désigne excluivement une méta-liste et X une méta-variable associée à une valeur compatible avec celle des éléments de A.

Exemples :

```
%A = '12', '48', '72', '56', 'X 135', 'PR' ;
%J = '72' ;
X = % RG SYMBOL (A, J) ; (est équivalent à X = 3 ;)
X = % RG SYMBOL (A, 56) ; est équivalent à X = 4 ;
X = % RG SYMBOL (A, PR) ; est équivalent à X = 6 ;

%B = 12, 48, 72, 56 ;
%K = 72 ;
%L = B ;
Y = % RG SYMBOL (L, K) ; (est équivalent à Y = 3)
Y = % RG SYMBOL (L, 56) ; (est équivalent à Y = 4)
```

REMARQUE :

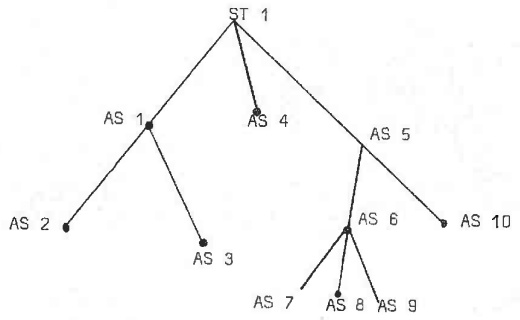
dans X = % RG SYMBOL (A, 56) ; 56 est traité comme une chaîne de caractères

dans Y = % RG SYMBOL (L, 56), 56 est converti en entier interne avant d'être comparé aux éléments de la liste L qui est identique à B

Y = % RG SYMBOL (L, 1007)␣P*T (S) serait équivalent à Y =␣␣P*T (S) ; car l'élément 1007 ne figure pas dans la liste L et on produit à la place un blanc supplémentaire.

Exemple :

Soit la structure



%J = % LH (ST 1, AS 4) équivalent à %J = 'AS 5'

{%K = % LV (ST 1, J) ; %K = % LV (ST1, AS 5)} équivalent à %K = 'AS 6'

%R = 'ST1'
%P = % LV (R, AS 6), % LH (R, P (1)), % LH (R, P (2))

Cette dernière méta-instruction équivaut à

%P = 'AS 7', 'AS 8', 'AS 9'
%S 7 = 'AS 7'
%P 4 = % NB PREDEC (ST 1, S 7) équivaut à %P 4 = 3
%P 7 = % NUM (P) équivaut à %P 7 = 3
C = % LH (ST 1, AS 2) - % NB PREDEC (ST1, AS 3) produira le texte C = AS 3 - 2

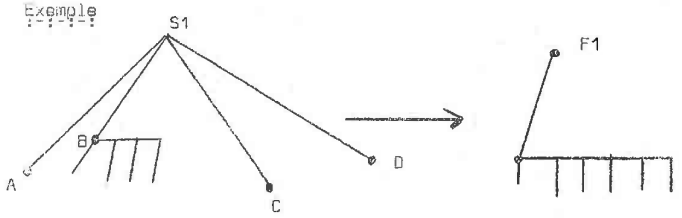
D'autres exemples seront vus dans le chapitre suivant : exemple de méta-modules et méta-fonctions.

6 - EXEMPLES D'ECRITURE DE META-MODULES

Exemple n° 1

Soit à transférer les valeurs des feuilles d'une structure vers les éléments d'une file fixe d'éléments de type simple. La structure émettrice peut avoir soit des feuilles de type simple, soit des feuilles de type file fixe d'éléments simples.

Exemple



S1 structure (A entier, B file (4) réel, C entier, D réel) ;
F1 file (6) réel ;

On analyse l'arborescence de la structure émettrice en recherchant des feuilles ; si une feuille est de type simple, elle est transférée, si elle est de type file, on transfère ses éléments les uns après les autres avant de passer à la feuille suivante ; on arrête le transfert quand le récepteur ou l'émetteur est "épuisé".

K est l'indice de la file réceptrice
I l'index du nombre de feuille de la structure émettrice
J l'indice courant des éléments des feuilles éventuelles de type file.

```
% DEF - MØD   STRUC FIL (S, F) ;
% LØCAL K, I, J ;
% K = 0 ;
% faire I = 1, % NBRE FEUIL (S) ;
% SI   % TYPE (% NØM FEUIL (S, I)) = { -16 = type des files fixes ;
                                      { Type F (*)
```



```
        % ALØRS      % X = NØM FEUIL (S, I) ;
                    % FAIRE J = 1 ,  % TAILLE (X) ;
                    % K = K +1 ;
                    % SI K  % TAILLE (F) ALØRS  ALLERA FINI ; % SINØN.
                        F (K) = X (J) de S ;
                    % FIN SI ;
                    % FIN FAIRE ;
        % SINØN      % K = K + 1 ;
                    F (K) = % NØM FEUIL (S, I) ;
        % FIN SI ;
        % FIN FAIRE ;
% FINI : FIN-MØD ;
```

Un exemple d'appel serait :

% STRUC FIL (S1, F1) ; où S1 et F1 sont précédemment déclarées ;

Le traitement de cet appel produirait le texte suivant

```
F1 (1) = 1 ;
F1 (2) = B (1) de S1 ;
F1 (3) = B (2) de S1 ;
F1 (4) = B (3) de S1 ;
F1 (5) = B(4)  de S1 ;
F1 (6) = C ;
  .
  .
  .
```

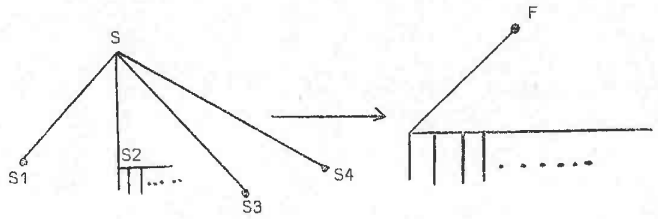
D ne sera pas affecté car F1 n'a que 6 éléments.

(*) L'utilisteur peut ne pas connaître explicitement la valeur des codes des types des différents objets CIVA. Pour cela, on pourrait définir quelques identificateurs de constantes réservées, placées dansù les tables de méta-objets et initialisées par le méta-processeur

Ainsi la variable TYPENT figurera dans la table des méta-objets avec la valeur -4. L'utilisateur peut donc directement utiliser ces méta-variables TYPENT, TYPREL, TYPBOL, TYPDECn TYPCAR, TYPFF(file fixe), TYPFM (File maximum), TYPFV (variable) TYPFIC (fichier) TYPSTR (structuré) etc...

Exemple n° 2

Transfert d'une structure comportant une file de taille variable vers une file réceptrice de taille variable.



Ce problème est très particulier, on suppose que la structure S a été définie comme un type dont la première feuille est simple, la deuxième feuille est toujours une file de taille variable, la troisième et la quatrième sont toujours simples.
On écrirait :

```
% DEF-MØD  TRANSFERT (S, F)
% LØCAL G ;
    .
    F (1) = % NØM FEUIL (S, 1) ;
    % G = % NØM FEUIL (S, 2) ;
    F (2,.) = G ;
    F (2 + taille actuelle de G) = % NØM FEUIL (S, 3) ;
    F (3 + taille actuelle de G) = % NØM FEUIL (S, 4) ;
    % FIN MØD ;
```

G a été défini pour éviter de réanalyser l'arborescence deux fois de suite pour déterminer le nom de la deuxième feuille de S.

Exemple n° 3

Transfert des feuilles d'une structure A vers celles de B qui ont

- même nom
- même prédecesseur (le même nom et le même nombre)

Ce méta-module est l'équivalent du MØVE-CORRESPØNDING de CØBØL ou MØVE BY NAME de PL/1.

On donnera deux textes d'écriture de ce méta-module qu'on appellera TRANSTR3

Dans le premier, on utilise que les méta-fonctions de base LH (lien horizontal), LV (lien vertical) ; et on n'analyse les arborescences des deux structures qu'une fois et une seule ;

On analyse parallèlement les deux structures et on ne s'intéresse qu'aux champs ayant le même nom dans chaque niveau.

On utilisera deux piles PA et PB (qui sont en fait des listes de méta-variables) pour décrire explicitement les arborescences .

Dans le second, l'analyse des structures n'est pas explicite, on fera usage de méta-fonctions spécifiquement adaptées (telles que NBRE FEUIL, NØM FEUIL, NØM CHAM, etc....).

L'écriture est plus directe, plus facile, mais très inefficace, car chacune des méta-fonctions citées, nécessitent une analyse complète ou partielle de l'arborescence, indépendamment de son contexte d'utilisation. Cette méthode est donc beaucoup moins rapide que la précédente, mais cette évaluation n'intervient qu'à la traduction, pas à l'exécution.

3.1 - Ecriture du méta-module TRANSTR3 à l'aide de LV et LH

```
% DEF-MØD  TRANSTR3 (A, B) ;
% LØCAL I, X, Y, PA, PB, EA, EB ;
% I = 1 ;
% X = % LV (A, A) ;
% Y = % LV (B, B) ;
% ET1 : PA (I) = X ; (*)
```

```
§ PB (I) = Y ;
§ EA = X ;
§ ET2 : EB = PB (I) ( )
          1              1       2
§ ET3 = SI  EA = EB  § ALØRS  § SI  § LV (A, EA) = 0 §§LV(B, EB) = 0
§ ALØRS § EA = EB ; § ALLERA ET4 ; § SINØN § SI LV (A, EA)≠ 0 §
   2                                              3
§ LV (B, EB) ≠ 0 § ALØRS § SI = I+1 ; § X = § LV (A,EA); §Y = §LV(B,EB) ;
                    3
                      3                    3          2
§ ALLERA ET1 ; § SINØN § ALLERA ET4 ; § FIN SI ; § FIN SI ;
§ SINØN § EB = § LH (B, EB) ; § SI EB = 0 § ALØRS § ALLERA ET4
   1
§ SINØN § ALLERA ET3 ; § FIN SI ; § FIN SI ;
§ ET4 : EA = § LH (A, EA) ; § SI EA ≠ 0 § ALØRS § ALLERA ET2 ;
§ FIN SI ;
§ ET5 : SI  I = 1 § ALØRS § ALLERA FINI ; § SINØN §I = I-1 ;
§X = § LH (A, PA (I)) ; § FIN SI ;
§ SI X = 0 § ALLERA ET1 ; § SINØN § ALLER A ET5 ; § FIN SI ;
§ FINI : FIN MØD ;
```

(*) Ne pas confondre une méta-instruction complète étiquetée avec une méta-instruction vide, suivie d'une instruction du langage.

Exemple :

```
1 §C = 25 ;
2 § ETIQ : P = C+2 ;
3 § ETIQ : ; A = C+2 ;
```

Les lignes 2 et 3 sont de nature différentes :

La deuxième est composée d'une seule méta-instruction ;

la troisième est composée d'une mété-instruction vide et d'une instruction du langage de base ; le point virgule servant de separateur.

Le texte généré après leur analyse est A = 25 + 2 ;

(il y a en plus dans la deuxième ligne de déclaration de la méta-variable P qui a pour valeur 27, et pouvant servir plus tard dans le texte-source.

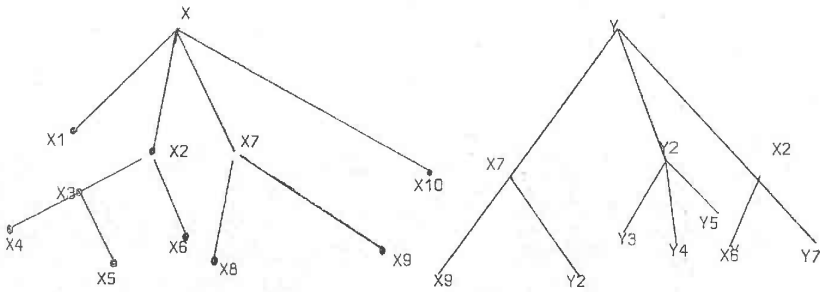
3.2 - Ecriture directe de TRANSTR3 sans analyse explicite

```
§ DEF - MØD TRANSTR3 (A, B)  (*)
§ FAIRE I = 2 , § NBRE FEUIL (A)
§ FAIRE J = 2,  § NBRE FEUIL (B)
§ SI § NØM FEUIL (A, I) = § NØM FEUIL (B, J) § §BREPRED(A,§NØMFEUI
                                                  (A,I)
= § NBRE PRED (B, § NØM FEUIL (B, J)  § ALØRS
§ X = § NØM FEUIL (A, I)
§ Y = § NØMFEUIL (B, J)
§ FAIRE   K = 1, § NBRE PRED (A, X) - 1
§ SI  § NBMPRED (A, X) = § NØM PRED (B, Y) § ALØRS X = §NØMPRED (A, X)
Y = § NØM PRED (B, Y) § SINØN § ALLERA  FINI
§ FIN FAIRE
```

(*) § NØM FEUIL (A, I) . de . A = § NØM FEUIL (B, J) . de . B
§ SINØN ; § FINI : FIN FAIRE ; § FIN FAIRE ;
§ FIN - MØD ;

3.3 - Exemple d'application de TRANSTR3

Soit X et Y deux structures représentées par les figures



On commence par comparer les éléments du niveau 1 entre eux
X1 de X ≠ X7 de Y ; X1 de X ≠ de Y2 de Y ; X1 de X ≠ X2 de Y
X2 de X ≠ X7 de Y ; X2 de X ≠ Y2 ; X2 de X = X2 de Y
On reprend à partir de X2 dans X et X2 dans Y
X3 de X2 de X ≠ X6 de X2 de Y ; X3 ≠ Y7
X6 de X2 = X6 de X2 de X ; X6 est une feuille alors on génère
X6 de X = X6 de Y ; on reprend à partir du niveau précédent après X2
X7 de X = X7 de Y ; X8 de X7 ≠ X9 de X7 ; X8 de X7 ≠ Y1 de X7
X9 de X7 de X = X9 de X7 de Y ; X9 est une feuille
on génère X9 de X = X9 de Y, on reprend à partir du niveau précédent
X10 de X ≠ X7 de Y ; X10 de X ≠ Y2 ; X10 de X ≠ X2 de Y ; fin d'exploration.

Exemple n° 4

Transfert conditionnel d'une structure B vers une autre structure A.

Problème :

Soit à écrire un méta-module qui ferait le transfert par le mot des feuilles d'une structure vers une autre avec des sous-conditions portant sur les valeurs de certains champs (ou feuilles) en paramètres

```
% DEF - MØD  TRANSCØND (A, B, CH1, CH2, CH3, CH4,.... CHN)

SI ("condition" (*) ) ALØRS
A = B ;
% FIN SI ;
% FIN - MØD
```

(*) "condition" désigne une expression booléenne quelconque portant sur les différents champs CHi en paramètres.

Exemple n° 5

Transfert de structure réalisant une fusion au sens de REIX (cf. 17)

Partant de deux structures A et B, on créée une nouvelle structure C, dont le mot des feuilles est la juxtaposition de celui de A et celui de B. Cette juxtaposition pouvant être conditionnelle ou non (la condition courante étant l'égalité de deux champs (feuilles) particuliers CHA et CHB appartenant respectivement à A et B.

```
% DEF-MØD  FUSIØN (A, B, C, CHA, CHB)
  %K = % NBRE FEUIL (A)
  SI  CHA = CHB  ALØRS
  % FAIRE I = 1, K ;
  % NØM FEUIL (C, I) = % NØM FEUIL (A, I) ; (cf Rq 2)
  % FIN FAIRE ;
  % FAIRE I = 1, NBRE FEUIL (B) ;
  % NØM FEUIL (C, K+ I) = % NØM FEUIL (B, I) ;
  % FIN FAIRE ;
  % FIN - MØD ;
```

REMARQUE :

Il est indispensable que la structure C soit convenablement décrite et déclarée pour recevoir l'ensemble des valeurs des feuilles de A et de B. On "remplit" ses feuilles sans se préoccuper de la façon dont on veut y accéder par la suite.

Exemple n° 6

Méta-module réalisant l'union de 2 ou plusieurs structures au sens de REIX (cf 17)

On veut faire le transfert des feuilles A vers celles de B qui ont le même nom... Si les deux structures ont le même nombre de feuilles chaque feuille de l'une ayant une feuille de même nom dans l'autre, on dira que la réorganisation est totale.
Si la structure réceptrice A a un nombre de feuilles inférieur à celui

de l'émettrice B, ayant feuille de A ayant une feuille de même nom dans B ; on dira qu'on réalise une extraction de structure ou partition

Dans le cas général, on parlera d'union de plusieurs structures pour en faire une seule ; on transfèrera les valeurs des feuilles des différentes structures B1, B2, B3..... vers les feuilles de même nom de la structure réceptrice résultante A cette union pouvant être faite en 1 ou plusieurs étapes.

a) Réorganisation d'une structure UNIØN (A, B)

On écrirait :

```
% DEF - MØD UNIØN 1 (A, B) (*)
% FAIRE I = 1, % NBRE FEUIL (A)
% FAIRE J = 1, % NBRE FEUIL (B)
% SI % NØM FEUIL (A, I) = % NØM FEUIL (B, J)
% ALØRS  % NØM FEUIL (A, I) de A = % NØM FEUIL (B, J)de B (*) (**)
% SINØN  % ALLER A RETØUR
% FIN SI
% RETØUR : FIN FAIRE
% FIN FAIRE
% FIN - MØD
```

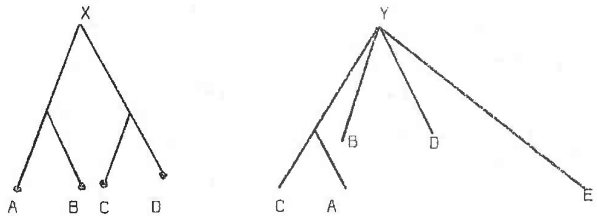
(*) Ce méta-module pour les mêmes raisons que pour les précédents est facile à écrire, mais nécessite un assez grand nombre d'analyses des arborescences des structures A et B.
On pourrait l'améliorer en déterminant dans un premier temps le mot des feuilles de A et de B sous forme des méta-liste dont les valeurs ont les noms des feuilles de chaque structure, puis de faire la recherche de celles qui ont le même nom.

(**) Rappelons qu'à l'intérieur d'un programme, chaque apparition d'une méta-fonction sera remplacée par sa valeur. La façon de traiter cette valeur dépend de l'instruction ou de la méta-instruction où est apparue cette méta-fonction ainsi si la valeur de % NØM FEUIL (A, I) = X et % NØM FEUIL (B, J) = Y, la méta-instruction % SI % NØM FEUIL (A, I) = % NØM FEUIL (B, J).....est une méta-instruction conditionnelle qui vaut la valeur . FAUX ., elle est évaluée à sa rencontre.

L'instruction % ALØRS % NØM FEUIL (A, I) = % NØM FEUIL (B, J).... générerait dans le cas supposé les caractères :
X = Y qui représentent une affectation dans le langage CIVA

Exemple d'utilisation

Soit deux structures X et Y représentées par leurs arborescences



àl'appel du méta-module UNIØN (X, Y)

1 - La méta-variable I varie depuis la valeur 1 à 4
(% NBRE FEUIL (X) = 4) , J variera depuis 1 à 5
on génèrera quand I = 1 et J = 2
A de X = A de Y
pour I = 2, J = 3
B de X = B de Y
pour I = 3 J = 1
C de X = C de Y
pour I = 4 J = 4
D de X = D de Y

b) Cas général d'union UNIØN (A, B1, B2, B3.....)

A est la structure réceptrice (résultante) ; les Bi sont des structures au nombre quelconque pour lesquelles il faut effectuer le transfert des feuilles ayant une homologue de même nom dans A

On pourrait utiliser le méta-module précédent UNION 1 et on

écrirait :

% DEF - MØD UNIØN (A,)
(ou bien % DEF - MØD UNIØN)

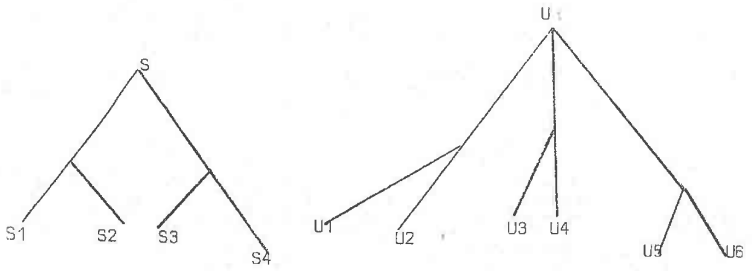
% FAIRE K = 2, % NUM (AF)
% mod UNIØN 1 (A, AF (K)) (ou UNIØN 1 (AF (1), AF (K))
% FIN FAIRE
% FIN - MØD

Exemple d'appel UNIØN (S, X, Y, U) ;
% NUM (AF) = 4 ; AF (1) = S ; AF (2) = X ; AF (3) = Y, AF (4) = Z ;

c) Méta-module d'union de plusieurs structures en précisant les noms des feuilles émettrices et les noms des feuilles réceptrices dans les paramètres.

Exemple :

Soient les arborescences suivantes :



% mod UNIØN 2 (U, S, (U3, S1), (U6, S2), (U4, S3))
Ce module effectuerait le transfert des valeurs des feuilles U3, U6, U4, de U vers les feuilles S1, S2, S3 respectivement de S
Le nombre de couples peut être quelconque
on écrirait :
% DEF - MØD UNION 2 (U, S) (ou UNIØN 2)
% FAIRE K = 3, % NUM (AF)
% AF (K, 1) de AF (1) = % AF (K, 2) de AF (2)
ou U ou S
% FIN FAIRE
% FIN MØD

Exemple n° 7

Exemple d'écriture de méta-fonctions

7.1 - Méta-fonction MIN (A, B)
% DEF - FØNCT MIN (A, B) ;
% SI A > B % ALØRS % MIN = A ; % SINØN % MIN = B ; % FIN SI;
% FIN - FØNCT ;

7.2 - MIN (A, B, C, D,.....)
% DEF - FØNCT MIN
% MIN = AF (1)
% FAIRE I = 2, % NUM (AF)
% SI AF (I) > MIN % ALØRS % MIN = AF (I); % SINØN ; % FIN SI ;
% FIN FAIRE ;
% FIN FØNCT ;

ORGANISATION GENERALE DU META-PROCESSEUR

Le méta-processeur analyse les textes sources de l'utilisateur (à partir d'un support magnétique disque ou bande) et produit un texte intermédiaire (ou chaîne codée) (également sur disque).

IL les analyse caractère par caractère, délimite les instructions ou méta-instructions : les premières sont immédiatement codifiées, les secondes immédiatement interprétées.

Ainsi au fur et à mesure de son analyse, le méta-processeur construit la table des méta-objets ainsi que celle des objets du programme origine.

Nous conviendrons d'appeler, dans la suite, codifieur, le module du méta-processeur qui codifie les instructions du texte source, propres au langage CIVA ; sa description détaillée ainsi que la gestion des tables des identificateurs est donnée dans (10).

Nous nous limiterons ici à la gestion des méta-objets et au traitement des méta-instructions.

Nous commencerons par étudier le mécanisme de définition et d'appel de méta-modules (ou de méta-fonctions).

1. - Définition des méta-modules et des méta-fonctions

Les textes (ou corps) des méta-modules ou de méta-fonctions seront reconnus au moment de leur définition c. a. d. à la rencontre de la méta-instruction Ø DEF - MØD <NØM> [(P1, P2,.....,PN)]
pour un méta-module et Ø DEF - FØNCT <NØM>[(P1, P2,...., PN)]
pour une méta-fonction

<NØM> est le nom de l'identificateur du méta-module ou de la méta-fonction

P1, P2 sont des paramètres formels ; ils suivent les mêmes règles que les identificateurs de méta-variables simples.

Ils peuvent être absents en totalité ou en partie ;
soit parce que les lignes de référence au méta-module ne comportent pas de paramètres effectifs correspondants ;
soit parce qu'ils seront repérés par la méta-fonction AF (AF (i) désignant le ième paramètre effectif au moment de l'appel ; à l'intérieur du corps

du méta-module.

Les corps des méta-modules sont délimités par la méta-instruction Ø FIN - MØD et ne doivent comporter que des méta-instructions ou des instructions entières.

Par contre, le corps d'une méta-fonction est délimité par la méta-instruction Ø FIN FØNC et ne doit comporter que des méta-instructions dont l'une au moins affecte une valeur à une méta-variable de même nom que la méta-fonction.

Une ligne de définition de méta-module ou de méta-fonction ne doit pas apparaître à l'intérieur d'une instruction ou d'une méta-instruction ; elle constitue à elle seule une méta-instruction complète. (Dans les macros générateurs plus complexes, (Ref 24), elle peut intervenir n'importe où dans le texte source).

Le corps d'un méta-module ou d'une méta-fonction peut contenir des définitions ou des lignes de références à d'autres méta-modules ou méta-fonctions.

Les méta-modules ainsi définis sont locaux au module qui les contient et ne sont "activés" que lors de l'appel à celui-ci.

Les textes de méta-modules (méta-fonctions) ne sont pas sauvegardés en mémoire. Ils sont directement repérés par leur adresse sur disque(numéro du bloc et adresse du premier caractère du texte à l'intérieur du bloc) dans la table des méta-modules.

On réservera un espace mémoire de la taille d'un ou plusieurs blocs du disque, qui servira de buffer pour les corps des méta-modules et des méta-fonctions. La recherche du texte du nom d'un méta-module consistera à regarder s'il est présent en mémoire sinon à l'y amener depuis le disque. On le chargera à la place du bloc le plus ancien dans le buffer des méta-modules. A la rencontre de Ø DEF - MØD ou Ø DEF - FØNC, le méta-processeur remet à jour la table des identificateurs de méta-modules.

2. - APPEL (ou REFERENCES) des META-MODULES ou des META-FONCTIONS

Une ligne d'appel à un méta-module ou à une méta-fonction a la forme suivante

Ø <NØM>[(P1, P2,....., PN)]

<NØM> désigne le nom de l'identificateur du méta-module appelé.
P1, P2,..... sont les paramètres effectifs. Ils peuvent être soit des identificateurs de méta-variables, soit plus généralement des méta-expressions comportant des appels de méta-fonctions, (en particulier des constantes entières ou chaînes de caractères).
La virgule sert de séparateur.

La correspondance paramètre effectif, paramètre formel se fait par la position respective dans les lignes de définition et d'appel.

La ligne de référence à un méta-module ne doit pas apparaître à l'intérieur d'une méta-instruction ou d'une instruction. Elle constitue une méta-instruction complète (c. a. d., elle se termine par un point virgule).

Par contre la ligne de référence à une méta-fonction peut apparaître dans n'importe quelle méta-expression ou n'importe où à l'intérieur d'une instruction. A la rencontre d'une ligne d'appel de méta-module, le méta-processeur commence par évaluer d'abord les paramètres effectifs, puis analyse successivement les instructions et les méta-instructions qui composent son corps. (IL y a éventuellement chargement préalable du bloc disque qui contient le corps du méta-module en question) ; les méta-instructions sont interprétées, les instructions sont codifiées et le résultat de la codification constitue la chaîne codée de sortie.

Au moment de l'analyse du corps d'un méta-module, il procède à la substitution des paramètres effectifs évalués aux paramètres formels. A la fin de l'analyse du texte, il reprend son analyse à la ligne située immédiatement après la ligne d'appel.

A la rencontre d'un appel de méta-fonction, le méta-processeur sauvegarde éventuellement le contenu du buffer d'interprétation avant d'évaluer les paramètres effectifs, puis d'analyser les méta-instructions du corps de la méta-fonction. A la fin de cette analyse, le méta-processeur génère la valeur de la fonction (qui peut être une valeur entière ou une chaîne de caractères quelconques à l'exclusion de caractères spécieux propres au méta-processeur tels que , , ; , % etc....) à l'endroit de l'appel avant de reprendre son analyse au caractère immédiatement situé après cet appel.

Comme les appels de méta-modules (ou de méta-fonctions) peuvent être imbriqués à un niveau quelconque, il est nécessaire de sauvegarder les informations constituant le contexte de l'apparition de chaque appel

et de les restituer après le traitement de celui-ci ; on utilisera pour cela une pile PILE CØNT.

Par ailleurs, l'appel d'une méta-fonction pouvant intervenir aussi bien dans une ligne du programme principal (extérieur à tout méta-module), d'un méta-module, ou dans un paramètre effectif, il est nécessaire de distinguer plusieurs modes de traitement ; mode d'évaluation ; mode d'analyse du corps d'un méta-module ou une méta-fonction ; mode de substitution de paramètres effectifs aux paramètres formels ; mode de définition de méta-module ou de méta-fonction et le mode de saut. On entre dans le mode d'évaluation au moment de la rencontre de %< NØM >.... ou NØM est une identificateur de méta-module ou de méta-fonction.

L'évaluation des paramètres effectifs se fait en déplaçant les paramètres constituant le nom du méta-module ou de la méta-fonction, suivis des caractères constituant chacun des paramètres effectifs ; y compris les virgules séparatrices vers une file des paramètres non évalués, si pendant ce déplacement on rencontre un appel de méta-fonction, on rentre dans une nouvelle évaluation.

La fin d'une évaluation se termine par la rencontre de la parenthèse fermante de la ligne d'appel.

Pour détecter celle-ci, un compteur décompteur de parenthèses pendant le mode d'évaluation seulement est nécessaire. A la fin d'une évaluation, on rentre obligatoirement dans le mode d'analyse du corps du dernier méta-module ou méta-fonction dont les paramètres (y compris le nom) sont sur la pile des paramètres non évalués.

Avant de rentrer dans ce mode, il est nécessaire de déplacer les paramètres complètement évalués vers une zone auxiliaire pour ne pas les détruire par des évaluations éventuelles rencontrées au moment de l'analyse du corps, ou même par le résultat des fonctions appelées. On supprime alors les informations de la pile des contextes concernant l'évaluation.

Dans le mode d'analyse du corps de méta-module ou méta-fonction, le texte source provient d'un corps de méta-module donc nécessite une recherche avec ou non lecture sur disque du bloc contenant le module en question.

Pour les méta-modules, il y a interprétation ou codification des instructions qui les composent ; donc la sortie de ce mode est toujours le buffer d'interprétation.

Pour les méta-fonctions, outre l'interprétation des méta-instructions qui les composent, il y a transfert (avec conversion éventuelle) du résultat soit vers la pile des paramètres non évalués si le mode est un mode d'évaluation, soit vers le buffer d'interprétation (après avoir rechargé dans le buffer d'interprétation l'instruction du sommet de la pile des instructions incomplètes qui ont été interrompues au moment de la rencontre des appels de méta-fonction) si le mode précédent dans la pile des contextes est un mode d'analyse ou si la pile est vide.

Dans le mode de substitution, les caractères d'entrée proviennent de la zone auxiliaire où sont sauvegardés les paramètres entièrement évalués. Ils sont toujours transférés dans le buffer d'interprétation.

Ce mode intervient au moment de la rencontre d'un paramètre formel ou l'appel de la méta-fonction AF et se termine au moment où l'on rencontre une virgule séparant les paramètres dans la zone auxiliaire ou un signe ≠ terminant le dernier paramètre effectif d'un même appel. (ce signe a été placé au moment du transfert des paramètres complètement évalués de la pile des paramètres non évalués vers la zone auxiliaire).

Le mode d'analyse du corps d'un méta-module ou de méta-fonction se termine à la rencontre de la méta-instruction Ø FIN - MØD ou Ø FIN - FØNC ; on restaure alors le contexte précédent dans la pile des contextes ; on supprime la nomenclature locale de la table des méta-variables et des méta-modules ou fonctions. On restitue la place des paramètres dans la zone auxiliaire.

Le mode définition de méta-module ou méta-fonction commence à la rencontre des caractères Ø DEF - MØD, ou Ø DEF - FØNC. On remet alors à jour la table des méta-modules. C'est toujours la plus récente version d'un module qui sera prise en considération en cas de double définition.

Dans ce mode, on saute tous les caractères suivants jusqu'à la rencontre de Ø FIN - MØD ou Ø FIN - FØNC correspondant.

Pour permettre des définitions de méta-modules ou de méta-fonctions locales, un compteur décompteur de Ø DEF - MØD et Ø FIN - MØD ou de Ø DEF - FØNC et Ø FIN FØNC est nécessaire.

Une fois, ce mode terminé, on l'efface de la pile des contextes et on restaure le précédent.

REMARQUE :

Quand la pile des contextes est vide (au début l'analyse d'un programme quelconque ou après restaurations successives de contextes) On est dans le mode d'analyse du programme principale de l'utilisateur, qu'on peut assimiler à un mode d'analyse particulier de corps d'un méta-module de nom implicite : PP et n'ayant aucun paramètre effectif Ce mode se terminerait par la fin de fichier sur disque (fichier constitué par le texte source d'un utilisateur) ou par la rencontre d'une instruction FIN indiquant la fin de traduction.

Le mode de saut commence après la reconnaissance d'une méta-instruction de saut inconditionnel ALLERA ETIQ ou ETIQ est le nom d'une étiquette en avant (ne figurant pas dans la table des méta-variables au moment de l'analyse de la méta-instruction en question).

A ce moment là le méta-processeur saute les caractères qui suivent cette méta-instruction jusqu'à la rencontre de l'étiquette citée. Dans ce mode il ne tient compte d'aucun caractère spécial comme la virgule, le point virgule, les parenthèses etc... Il ne tient compte que des méta-instructions Ø FAIRE, Ø FIN FAIRE, Ø DEF - MØD, Ø DEF + FØNC Ø FIN - MØD , Ø FIN - FØNC (voir traitement de la méta-instruction Ø ALLERA dans le chapitre suivant) et des étiquettes des méta-instructions Ce mode de saut se termine à la rencontre de l'étiquette recherchée. Il est alors effacé de la pile des contextes et on restaure le contexte précédent.

Il y a un autre mode qu'on pourrait appeler mode d'exécution qui intervient après l'évaluation complète des paramètres effectifs d'une méta-fonction de BASE (AF, NUM, LV, LH, etc...) qui sont des procédures écrites en langage machine. Mais avant de commencer leur exécution, on doit procéder à la transmission convenable des paramètres. On assimilera ce mode à un mode particulier d'analyse de corps de méta-modules et les informations concernant son contexte seront les mêmes ; toutefois la manière d'acquérir le résultat diffère légèrement.

Avant de donner les spécifications précises de chaque mode, rappelons l'ensemble des zones utilisées pendant le traitement d'un appel.

- Pile des contextes PIL CONT avec les différents modes et leurs informations
- Pile des paramètres non évalués (PIL Par) où seront rangés les paramètres au fur et à mesure de leur évaluation
- Pile des instructions incomplètes (PIL INC) où seront rangées les parties d'instructions interrompues par un appel de méta-fonction
- Pile auxiliaire (PIL EV) où seront sauvegardées les paramètres une fois qu'ils sont complètement évalués
- Buffer d'interprétation où seront rangées successivement les instructions ou méta-instructions dès qu'elles sont constituées ; si ce sont des méta-instructions, elles sont interprétées directement dans ce buffer ; si ce sont des instructions elles sont transférées auparavant vers le buffer de codification. Ce transfert s'accompagnant de la substitution complète de toutes les méta-variables qui existeraient encore dans ces instructions, séparerait ainsi le traitement du codifieur et du méta-processeur.
- La table des noms des méta-modules et des méta-fonctions qui évolue comme une pile (cas des définitions de modules locaux à d'autres)
- La table des méta-variables, étiquettes qui elle aussi, évolue comme une pile à cause des nomenclatures locales à chaque module.

Pour les piles PIL INC, PIL PAR, PIL EV, les éléments sont des chaînes de caractères de longueurs différentes ; chaque chaînede caractères sera repérée par le pointeur de son début et un signe spécial qui indiquera sa fin.

Pour la pile PIL CØNT, les informations qu'elle contient dépendent du mode dans lequel on travaille.

Pour le mode d'évaluation, il y a d'abord son code.

La valeur du pointeur du premier caractère libre dans la pile non évalués PPAR

La valeur du pointeur du premier caractère libre dans la pile des instructions incomplètes dans le cas où ce mode d'évaluation a interrompu une instruction en cours de formation dans le buffer d'interprétation. Cette valeur servira plus tard à restaurer la valeur d'une fonction à l'endroit d'interruption.

Pour le mode d'analyse de corps d'un méta-module ou d'une méta-fonction, les informations à sauvegarder sont les suivantes :

- le code de ce mode (AN)
- le pointeur vers le nom du méta-module ou de la méta-fonction dans la pile des paramètres complètement évalués
- les pointeurs respectifs vers le début des paramètres effectifs évalués
- le nombre de ces paramètres
- le pointeur du premier mot libre dans la table des noms des méta-modules et fonctions (ceci pour effacer les noms des méta-modules locaux qui seront définis pendant le traitement d'un appel)
- le pointeur du premier mot libre dans la table des méta-variables (pour supprimer la nomenclature locale après la fin de ce mode
- le pointeur du texte d'entrée d'où on vient

On pourrait aussi empiler le nombre des éléments de la table des noms de méta-modules

3 - Interprétation des Méta-Instructions

3.1 - Rappel du processus de méta-traitement

Quand le méta-processeur est en mode analyse de corps de méta-module ou de méta-fonction (y compris le programme principal), il transfère les caractères les uns après les autres dans le buffer d'interprétation jusqu'à la rencontre d'un point virgule. Ce dernier délimite alors une méta-instruction ou une instruction, si le premier caractère est un %, il s'agit d'une méta-instruction ; sinon il s'agit d'une instruction ; dans ce dernier cas, il y a transfert du contenu du buffer d'interprétation vers un autre buffer dit de codification. Ce transfert ayant pour objet de remplacer toutes les méta-variables éventuelles de l'instruction par leurs valeurs.

Pour toutes les méta-instructions, il y a un traitement éventuel des étiquettes si elles existent.

3.2 - Méta-Instruction d'affectation

Le méta-processeur interprète directement l'expression contenue dans le buffer (signe = compris) en utilisant une méthode d'évaluation des expressions arithmétiques en infixée (29). Si l'opérande de gauche ne figure dans la table, il l'y rajoute en même temps que la valeur et le type du résultat de l'évaluation de l'expression de droite. (Les règles citées dans le paragraphe "Méta-expressions" s'appliquent pour le type et les conversions).

3.3 - Méta-instruction de SAUT : Ø ALLERA

Le méta-langage CIVA autorise des sauts inconditionnels vers une ligne de texte source en "avant" ou en "arrière". Rappelons que quand le méta-processeur rencontre une étiquette d'une méta-instruction, il range l'identificateur correspondant dans la table des méta-variables avec son type "étiquette", l'adresse du caractère qu'elle repère dans le texte source (adresse relative par rapport au début du bloc d'entrée-sortie) et le numéro du bloc où figure cette étiquette sur le disque.

Dans les méta-modyles, les étiquettes sont locales.

a) Etiquette en arrière

Si on analyse le corps d'un méta-module, la recherche de l'étiquette se limite à la nomenclature locale du module ;

Si on analyse le programme principal, on effectue un test de présence en mémoire du bloc associé à l'étiquette ; s'il n'y est pas il y a changement depuis le disque du bloc en question vers le buffer d'entrée-sortie.

On reprend alors l'analyse dans les deux cas à partir du premier caractère repérée par l'étiquette.

b) Etiquette en avant

Si on analyse le corps d'un méta-module, la recherche de l'étiquette en mide de saut est limitée au corps du méta-module si elle est absente il y a émission d'un message d'erreur.

Si on analyse le programme principal, on travaille alors en mode de saut jusqu'à la rencontre de l'étiquette source à l'exception du point virgule, du Ø.

Dans le mode de saut, le méta-processeur repère les paires de méta-instructions (Ø DEF - MØD, Ø FIN - MØD), (Ø DEF-FØNC, Ø FIN-FØNC), (Ø FAIRE, Ø FIN FAIRE).

Quand il rencontre la première méta-instruction d'une de ces paires, il suspend sa recherche jusqu'à la rencontre de la deuxième en ignorant tous les caractères rencontrés. Il ne reconnaît pas ainsi les étiquettes situées à l'intérieur des méta-modules et des boucles d'itération.

Rappelons qu'il n'est pas autorisé d'écrire une méta-instruction ALLERA à l'intérieur d'un méta-module vers une ligne située à l'extérieur ; en effet, le méta-processeur rencontrerait Ø FIN - MØD (ou Ø FIN - FØNC) sans avoir rencontré Ø DEF - MØD (ou Ø DEF - FØNC) correspondant.

3.4 - Méta-instruction conditionnelle Ø SI.....Ø ALØRS.....Ø SINØN....Ø FIN SI

Les méta-instructions conditionnelles peuvent être imbriquées à un niveau de profondeur quelconque.

Pour les interpréter, on les décompose de la façon suivante :

- On délimite la méta-instruction qui commence par Ø SI et qui se termine par Ø ALØRS qu' on assimile à un point virgule terminateur ; suivant le résultat de l'interprétation de cette méta-instruction, soit ont analysera ensuite les instructions et les méta-instructions dont la première commence derrière Ø ALØRS et la dernière se termine par un point virgule explicite ou par Ø SINØN qui sera assimilé à un point virgule terminateur ; soit on "sautera" jusqu'à l'instruction Ø SINØN correspondante.

- On délimite alors la méta-instruction Ø SINØN qui fait à elle seule une méta-instruction complète qu'on interprétera immédiatement selon le résultat de cette interprétation ; soit on analysera les instructions et les méta-instructions dont la première commence

derrière Ø SINØN et dont la dernière se termine soit par un point virgule explicite , soit par la rencontre du Ø FIN SI correspondant ; soit on "sautera" jusqu'à ce Ø FIN SI.

- on délimite alors la méta-instruction Ø FIN SI qui constitue à elle seule une méta-instruction complète qui sera immédiatement interprétée.

- L'analyse reprend derrière le Ø FIN SI.

Pour permette l'imbrication à un niveau quelconque des méra-instructions conditionnelles,les modules qui les traiteront, utiliseront une pile de travail appelée PILE SI,

Nous désignerons par N le nombre de niveaux des SI imbriqués

Il est initialisé à 0 au début du programme ; il est incrémenté de 1 à chaque rencontre de la méta-instruction SI ; décrémenté de 1 à chaque rencontre de Ø FIN SI.

X sera un indicateur égal à 0 au début du programme. Il est positionné à un à la rencontre de chaque Ø SI et à 0 quand on est à l'extérieur de toute paire SI-FINSI.

VAL désigne le résultat de l'expression qui suit immédiatement Ø SI.

a - Interprétation de Ø SI <expression> ; ou Ø SI <expression> Ø ALØRS

Le méta-processeur positionne X à 1, incrémente N de 1 ; évalue l'expression qui suit Ø SI ; empile la valeur de N et de VAL.

Il teste alors la valeur de VAL ; si Val est $>$ 0 ou . VRAI. l'interprétation est terminée ; (c. a. d. la prochaine instruction ou méta-instruction à analyser est celle qui suit Ø ALØRS).

si VAL est $\leqslant$ 0 ou . FAUX.

Le méta-processeur "saute" jusqu'au prochain Ø SINON de même nombre niveau. Dans ce mode de SAUT comme pour la méta-instruction ALLER A, il recherche les paires de Méta-instructions Ø SI - Ø FIN SI

Pour chaque Ø SI N est incrémenté de 1
Pour chaque Ø FIN SI, il est décrémenté de 1

b- Interprétation de Ø SINØN ;

Le méta-processeur compare la valeur du deuxième sommet de la pile PILE SI (valeur de N au dernier Ø SI rencontré) avec la valeur actuelle de N.

S'ils sont différents, il y a message d'erreur dûe à un défaut d'imbrication : (Le nombre de SI rencontrés est différent du nombre de SINØN ou de FIN SI).

S'ils sont égaux, alors deux cas peuvent se présenter

1 - La valeur du sommet de la pile (valeur de l'expression) est 0 ou . VRAI.

Le méta-processeur "saute" comme il a été dit précédemment la méta-instruction Ø FINSI correspondante de même niveau.

2 - La valeur du sommet de la pile PIL SI et 0 ou . FAUX. l'interprétation de la méta-instruction est terminée.

c - Interprétation de Ø FIN SI ;

1 - Si la valeur du deuxième sommet de la pile est égale à la valeur de N, alors on retire les deux sommets de la pile ; on décrémente N de 1 ; si N = 0 (pile vide) on se retrouve à l'exérieur de toute paire Ø SI - Ø FIN SI, on réinitialise l'indicateur X à 0 et l'interprétation est erminée ; si N $\neq$ 0, l'interprétation est terminée.

2 - Si la valeur du deuxième sommet de la pile est différente de celle de N ; décrémente N de 1 ; si la pile est vide on réinitialise X à 0 sinon l'interprétation est terminée (ceci arrive quand le méta-processeur a opéré en mode de saut et que pendant ce mode de Ø SI ont été rencontré mais non interprétés).

3.5 - Interprétation de Ø FAIRE V = e1, e2, e3 ; et Ø FIN FAIRE

e1, e2, e3 sont de méta-expressions quelconques ; V est un identificateur de méta-variable. Les méta-instructions Ø FAIRE peuvent être imbriquées à un niveau quelconque ; pour permettre cette imbrication, on utilisera une Pile de travail PIL FAIRE qui sera mise à jour à chaque Ø FAIRE ou Ø FIN FAIRE

1 - Ø FAIRE expression

L'interprétation de cette méta-instruction se fait de la façon suivante

- on effectue la méta-affectation ØV = e1
- on empile l adresse de V et la valeur de l'expression e2
- on empile la valeur de e3 (si e3 est absent, elle est égale à 1).
- on empile l'adresse de retour qui est l'adresse du premier caractère après e3.

2 - Ø FIN FAIRE

Pour Ø FIN FAIRE, on procède de la façon suivante

- on effectue la méta-affectation ØV = V + e2, (e2 étant le 3ème sommet de la pile PILE FAIRE ; V étant le 4ème sommet de la pile
- on teste si V est inférieur ou égal à e3 (1er sommet de la pile) si oui, on se branche à l'adresse située au 1er sommet de la pile, sinon on retire les 4 premiers sommets de la pile ce qui termine l'interprétation de cette méta-instruction Ø FIN FAIRE.

4 - ETUDE DETAILLEE DU META-TRAITEMENT

4.1 - Formation des instructions et méta-instructions

Quand il travaille en mode d'analyse de corps de méta-modules (y compris le texte du programme principal), le méta-processeur lit le texte d'entrée caractère par caractère ; excepté les cas des symboles spéciaux tels que ; Ø , (,) , . Il les transfère un à un dans le buffer d'interprétation ; (on peut supprimer les blancs inutiles du texte pendant cette recopie).

On arrête le transfert à la rencontre d'un terminateur d'instruction ou de méta-instruction (exemple ";" ; Ø ALØRS, Ø SINØN, Ø FIN SI, Ø FIN-MØD, Ø FIN-FØNC, Ø SINØN,..... peuvent servir de terminateur en l'absence de point virgule explicite).

Puis on analyse le contenu du buffer d'interprétation :

si le premier caractère est un Ø, il s'agit d'une méta-instruction : elle est alors interprétée (voir le chapitre précédent) ; avec éventuellemnt le traitement des étiquettes si elles sont présentes ; sinon il s'agit d'une instruction ; on transfère alors les caractères les uns après les autres depuis le buffer d'interprétation vers le buffer de codification.

Pendant ce transfert, on délimite les identificateurs pour reconnaître les méta-variables qu'on remplace alors par leur valeur.

Une fois le transfert terminé, on codifie l'instruction ainsi formée pour produire la chaîne codée correspondante, on réinitialise les buffers d'interprétation et de codification et on reprend l'analyse dans le texte d'entrée.

4.2 - Analyse du caractère Ø <NØM>

Si <NØM> désigne un nom de méta-instruction et non un identificateur de méta-module ni de méta-fonction, le caractère Ø est transféré comme les autres dans le buffer d'interprétation ; (c'est lui qui permettra de reconnaître par la suite une méta-instruction d'une instruction). Dans ce cas, la rencontre de Ø sert éventuellement de terminateur d'instruction

ou de méta-instruction avant son transfert proprement dit.

Si ⟨NØM⟩ identifie un méta-module on rentre dans le mode d'évaluation rentre dans le mode d'évaluation.

Si ⟨NØM⟩ identifie une mét-fonction (l'appel %⟨NØM⟩ apparaît donc à l'intérieur d'une instruction ou méta-instruction dont on a déjà transféré les caractères précédents cet appel dans le buffer d'interprétation). Alors On commence par sauvegarder le contenu (partiel) du buffer d'interprétation dans la pile PIL INC des instructions interrompues (donc incomplètes) par un appel de méta-fonction ;

On regarde la nature du mode précédent dans la pile des contextes : si celui-ci était un mode d'analyse (ou si la pile est vide), le résultat de la fonction doit être directement envoyé vers le buffer d'interprétation après chargement de l'instruction interrompue depuis PIL INC ; on sauvegardera alors le pointeur du début de cette instruction dans PIl INC, pour pouvoir la recharger plus tard dans le buffer ; les différentes instructions ou méta-instructions incomplètes sont séparées entre elles par un signe spécial (ex : #).

Si le mode précédent est un mode d'évaluation, le résultat de la fonction fait partie d'un paramètre effectif ; il devra donc être transféré dans la pile des paramètres non évalués PIL PAR ; on sauvegardera donc le pointeur du premier caractère libre de cette pile.

Rappelons que le mode d'évaluation nécessite les informations suivantes :

le code d'évaluation ; le pointeur vers le premier caractère libre de la pile PIl Par des paramètres non évalués avant le début de l'évaluation (pour le déplacement des paramètres et pour une libération ultérieure après évaluation complète).

Pendant l'évaluation des paramètres effectifs, il est indispensable de reconnaître les parenthèses fermantes de fin d'appel.

Pour cela, on disposera d'un compteur décompteur de parenthèses N PARENT initialise 0 au départ et remis à 0 chaque fois que la pile des contextes est vide.

Dans les modes d'évaluation N PARENT est incrémenté de 1 pour chaque parenthèse ouvrante, décrémenté de 1 pour chaque parenthèse fermante.

La valeur de N PARENT est sauvegardée dans la pile des contextes pour la première parenthèse qui suit %⟨NØM⟩ ;

4.3 - Analyse de la parenthèse ouvrante (

Si le méta-processeur travaille dans le mode d'évaluation, alors il incrémente N PARENT de 1. Pour la première parenthèse ouvrante de ce mode précédent le premier paramètre effectif, il sauvegarde en plus la valeur de N PARENT dans la pile des contextes.

Si le méta-processeur travaille dans tous les autres modes, il traite cette parenthèse comme tous les autres caractères.

4.4 - Analyse de la parenthèse fermante)

a - On est dans un mode autre que l'évaluation
On traite la parenthèse fermante comme les autres caractères

b - On est dans un mode d'évaluation
On compare la valeur de N PARENT avec sa valeur sauvegardée

Si elles sont différentes on décrémente N PARENT de 1 et passe au caractère suivant dans le texte d'entrée.

Si elles sont égales l'évaluation est terminée. On décrémente N PARENT de 1 ; on commence par déplacer les paramètres ainsi entièrement évalués vers la pile PIL EV (on connaît le pointeur du début de ces paramètres dans la pile PIL PAR : il est dans la pile des contextes.

On connaît le pointeur du dernier caractère, c'est celui de la parenthèse fermante de fin d'évaluation).

Rappelons que dans le mode d'évaluation, le nom du méta-module ou de la méta-fonction appelée est considéré comme un paramètre).

On restaure la valeur du pointeur du premier caractère libre dans PIL PAR

On supprime alors le mode d'évaluation et ses informations du sommet de la pile des contextes ; et on se prépare à entrer dans le mode d'analyse du méta-module ou méta-fonction correspondante ; dont les informations sont les suivantes :

- Le code du mode d'analyse ;
- les pointeurs vers le nom et les paramètres effectifs dans PIL EV ;
- le nombre de ces paramètres ;
  (ces dernières informations recueillies au moment du déplacement des paramètres après leur évaluation vers PIl EV).
- la valeur du pointeur PE du texte d'entrée ;
- la valeur du nombre effectif de méta-fonctions et méta-modules effectifs (pour la suppression des méta-modules ou méta-fonctions locales qui seront éventuellement définies pendant le traitement de l'appel en cours).
- la valeur du pointeur PS de sortie ; recueilli d'après les informations du mode précédent (qui désigne soit l'adresse du premier caractère de l'instruction incomplète du sommet de la pile PIL INC, soit l'adresse du premier caractère libre dans la pile PIL PAR et qui servira à la génération du résultat des méta-fonctions. Pour les méta-modules ce pointeur n'est pas nécessaire. Il est toujours égal à l'adresse de début du buffer d'interprétation.Puisqu'un module ne peut pas interrompre, ni une méta-instruction, ni une instruction, il ne peut pas intervenir dans un paramètre effectif, en plus, il ne produit pas de résultat).
- le pointeur du premier caractère libre dans la pile PIL EV pour la restauration éventuelle de l'espace occupé par les paramètres effectifs des appels intervenant dans le corps du méta-module ou méta-fonction en cours d'analyse ;
- le pointeur du premier mot libre de la table des méta-variables (pour le début de la nomenclature locale et sa restauration après l'appel).

4.5. - Analyse de Ø FIN - MØD

A la rencontre de Ø FIN - MØD, le méta-processeur supprime le mode d'analyse en cours du sommet de la pile et restaure le contexte du mode précédent.

4.6 - Analyse de Ø FIN - FØNC

Avant de restaurer le contexte précédent dans PIL CONT, le méta-

processeur doit générer la valeur du résultat de la fonction à l'endroit de l'appel. Le nom de la fonction qui identifie également une méta-variable locale est repéré par son pointeur dans la pile des contextes ; grâce à la valeur de PS (pointeur de sortie), on générera la valeur du résultat de la fonction soit directement dans la pile PIL PAR, soit dans le buffer d'interprétation après avoir chargé du contenu de l'instruction au sommet de la pile PIL INC.On efface ensuite le contexte en cours de la Pile des contextes.

4.7 - Correspondance entre paramètres formels et effectifs

Les paramètres formels ne sont traités (délimités) qu'au moment du mode d'analyse d'un corps de méta-module ou de méta-fonction.

On les adjoint alors à la table des identificateurs locaux associée au module en cours d'analyse ; cette table est formée par les noms des paramètres formels et les identificateurs déclarés dans la directive Ø LØCAL si elle existe.

On établit la correspondance paramètre formel - paramètre effectif par l'intermédiaire des pointeurs situés dans la pile des contextes vers la pile PIL EV des paramètres entièrement évalués. On rentre dans le mode de substitution soit :

- à la rencontre de Ø AF (i)
- pendant l'interprétation des méta-instruction s ou pendant la formation des instructions à codifier, à la rencontre d'un identificateur de paramètre formel

L'appel Ø AF (I) sera remplacé par la valeur du Ième paramètre effectif (soit à partir de la pile des contextes, soit à partir de la sous-table locale qui vient d'être décrite dans le mode de substitution, le pointeur d'entrée de texte à analyser PS pointe sur les caractères de la valeur d'un paramètre effectif dans PIL EV.

Remarque :

L'évaluation des parametres effectifs ne consiste pas à les calculer pour obtenir un seul résultat ; mais elle consiste à évaluer uniquement tous les appels de méta-fonctions qu'ils peuvent éventuellement contenir ; les autres caractères restant identiques à eux-mêmes. Dans PIL EV, les paramètres

effectifs seront séparés entre eux par un signe spécial (ex) ; ainsi la rencontre de ce signe termine le mode de substitution en cours et le supprime de la pile des contextes.

4.8 - Format des tables de méta-variables

Chaque méta-variable y est essentiellement représentée par son identificateur, son type en cours (une méta-variable a le type de la valeur qui lui est affectée : entier, chaîne de caractères, étiquette, liste), sa valeur ;

4.9 - Format de la table des noms de méta-modules et de méta-fonctions

A chaque identificateur de cette table est associé 1. Un indicateur (soit de méta-module, soit de méta-fonction de Base écrite en langage machine, soit méta-fonction d'utilisateur écrite en CIVA).

2. Le numéro du bloc qui contient le corps correspondant sur disque pour son chargement en mémoire

3. L'adresse relative du premier caractère dans ce bloc.

CONCLUSION

Nous nous proposons de situer le méta-processeur que nous venons de décrire dans le cadre du projet CIVA parmi l'ensemble des méta-processeurs existant actuellement.

Le méta-traitement en général peut être défini comme la trasformation préliminaire d'un texte source contenant des éléments dont la signification n'est pas définie au niveau du langage source en un texte source proprement dit du compilateur dont tous les éléments sont définis au niveau du compilateur

La macro-génération et la macro-substitution sont des formes de méta-traitement longtemps appliquées dans les langages d'assembleur. Leur introduction dans les langages évolués a été faite pour la première fois par Mc ILROY (Macro instruction extension ofcompiler langages CACM 1960).

L'utilisation d'un méta-traitement soulève trois problèmes :

- Le méta-langage doit-il avoir la même syntaxe que le langage de base ?

En principe, ceci semble souhaitable pour les programmeurs et les constructeurs de compilateurs.

- Le méta-processeur doit-il se réduire à traiter des textes en tant que chaînes de caractères ou être un véritable analyseur syntaxique capable de réaliser les substitutions au niveau des éléments syntaxiques.

- Le méta-processeur doit-il être indépendant du compilateur ou être partiellement intégré à celui-ci ?

Comme dans le méta-langage CIVA, une des tendances actuelles des nouveaux langages de Programmation actuels est la possibilité de définir dynamiquement de nouveaux types d'entités. Les structures dynamiques de PL/1 et les enregistrements d'Algol sont des exemples de cette nouvelle tendance.

Ayant défini de nouveaux types d'entités, le problème se pose alors de définir les opérations pour ces nouveaux types en utilisant les mêmes symboles opératoires que pour les quantités de type élémentaire.

On peut choisir de réaliser ces définitions en utilisant le concept de procédure (ou module) qui est un mécanisme simple de définition dynamique.

L'utilisation des procédures présente néanmoins l'inconvenient d'une faible efficacité au niveau de l'exécution : le concept de procédure est très puissant et très élégant mais son utilisation en toute généralité est très chère en exécution.

L'autre façon de définir dynamiquement des opérations utilise le méta-traitement.

Ce que nous avons réalisé ici dans le méta-langage CIVA avait pour point de départ l'opération d'affectation ; mais ceci reste valable pour toutes les autres opérations (addition, multiplication,....), et aussi au niveau des opérations d'entrée-sortie (acquisition de données dans le projet CIVA).

Certains langages prévoient des des macro-extensions syntaxiques pour des instructions autres que celles citées ci-dessus exemple définition d'une instruction "POUR" de ALGOL dans un langage qui n'en possède pas.

Les principaux langages évolués possèdent des phases de méta-traitement sont étudiés dans Réf. 4 - Réf. 8 - Réf. 6.

BIBLIOGRAPHIE

---

1 - Métasymbol CII 10070 ; manuel d'utilisation

2 - RUSSEL E. C. Méta-5 Manual version for SIGITA 7 (or CII 10070) under BPM

3 - R. J. CHEVANCE (CII - MACPRØ Deux concepts de macro processeurs

4 - Introduction to Compile Time facilities of PL/1 student text : form C 20 1689-0 IBM coporation 1968

5 - Macro LPS (CII

6 - Leavenwork B. M. Syntax Macros and Extended Translation Communication ACM 9.11 (Nov. 66)

7 - Cheatham T. E. Introduction of Definitional facilities into higher level Programming langages.

8 - Leroy H. A. A proposal for macro facilities in ALGOL ALGOL Bulletin n° 22 (revues 66)

9 - William KENT International Business Machines Corporation San Jose California Comm Computer Survey (Décembre 69)

10 - Melle PERROT Codification et gestion des table Etude lexicographique dans le projet CIVA Thèse soutenue à NANCY

11 - Cours de compilation de Monsieur PAIR Faculté des Sciences de NANCY
présenté à l'Ecole d'Eté d'Informatique de NEUCHATEL

12 - Chabrier — Acquisition de données dans le projet CIVA
(Thèse de spécialité à NANCY)

13 - Melle LION — Implantation des files dans le projet CIVA
Thèse à paraître (NANCY)

14 - SCHULTZ — Traduction des "Pour chaque" dans le projet CIVA
DEA 71 - 72) Documentation interne

15 - DERNIAME J. C. — Introduction au projet CIVA - Documentation interne

16 - REIX R. — Analyse en Informatique de gestion Tome 1 - DUNOD 71

17 - BAZERQUE B. — Implantation des objets du langage PL/1 sur une calculatrice

18 - DUCLOY J. — Compilation dans CIVA - Thèse de Docteur Ingénieur soutenue à NANCY le 10 Mars 1973

19 - DERNIAME J. C. — Traitements séquetiels des files et des ensembles, calculs Itératifs - Documentation interne (NANCY)

20 - Manuel d'utilisation FORTRAN IV CII étendu 3709 E FR SIRIS 7/SIRIS 8

21 - BROWN P. J. — A survey of Macro - review in AUTOMATIC Programming VOL. 6 Pergamon Press OXFORD and NEW-YORK

22 - IRONS E. T. — Expérience with an extensible langage
Communication ACM 13, 1er Janvier 1970
Procedures of the extensibles langages symposium
BOSTON May 69 Communicat. ACM Juil. 72

23 - WEGNER P. — Programming langages
Informations structures and Machines Organisation

24 - STRATCHEY C. — A general Purpose Macro Generator ;
Computer Jal Octobre 1965

25 - WAITE W. — A langage independant Macro processor ; Comm. ACM JUIL 67

26 - ROSEN S. — The compiler Building System Developed by Brooker and Morris
Programming Systems and langages
Mac Graw Hill Book Company N. W. 67

27 - Structures d'Informations - Cours de Monsieur PAIR Informatique NANCY

28 - CII - Manuel d'utilisation Fortran IV sous SIRIS 7/SIRIS 8
(Paragraphe 1.9.1.3.)

29 - BOLHET — Thèse soutenue à Grenoble Juin 67 Notations et Processus de traduction des langages symboliques.

30 - David GRIES — Compiler Construction for digital Computers
Cornell University

31 - DEBRAINE — Machine de traitement de l'information
Masson Compagnie

32 - J. du ROSCOAT — Conception de la programmation des ordinateurs
Masson et Compagnie.

NOM DE L'ETUDIANT : BENAMCHAR Lahoucine

NATURE DE LA THESE : DOCTEUR INGENIEUR

Numéro C.N.R.S : A0 8.725

VU, APPROUVE

& PERMIS D'IMPRIMER

NANCY, le 20.6.73

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

J.R. HELLUY