LABORATOIRE D'INFORMATIQUE THEORIQUE
& APPLICATIONS DE MARSEILLE
L. I. T. A. M.
DEPARTEMENT DE MATHEMATIQUE-INFORMATIQUE
LUMINY
UNIVERSITÉ AIX-MARSEILLE II

ISSN 0291 - 5413

INFORMATIQUE
FONDAMENTALE
ET
APPLICATIONS

Sommaire

-Mars 1985-

ADRESSE POSTALE : FACULTÉ DES SCIENCES DE LUMINY
MATHÉMATIQUE-INFORMATIQUE - LITAM - BAT. TPR2 - 9ème ETAGE
CASE 901 - 70, ROUTE LÉON LACHAMP - 13288 MARSEILLE CEDEX 9 - ☎ (91) 41.01.40 POSTE 3294 - 3110

Editorial

e. bianco

Chanson de Printemps

L'hiver fut long et rude.Mais voila que les camions fleurissent sur les routes,que l'air tiède et parfumé de mazout excite nos poumons.Dans l'azur,le doux vrombissement des avions de combat nous fait frissonner au doux réveil de la nature.

Le bleu de la mer,moiré d'huiles lourdes,baigne mollement les grandes carcasses ferrugineuses,et le vent du large ramène les puissantes senteurs extatiques: les Emirats du Golfe,le Texas riche et poivré,l'Arabie Saoudite...

L'oeuil égaré,le mouvement saccadé,le citadin règle son corps au rythme de la ville.Et le sourd grondement qui monte de la terre,martèle la pensée du pêcheur solitaire,et dans un lent crescendo prépare le réveil de la meute endormie.

Le sang nouveau coule doucement dans les aiguilles de la transfusion,répandant dans le monde la richesse des pauvres,et comme l'écume brillante accompagne la vague qui déferle,dans le gai sida miroite la sensation du ferment exotique.

Le citoyen épuisé par la saison sinistre,regagne peu à peu le goudron verdoyant des campagnes aseptisées,desherbées,heureux de retrouver l'air vif qui fait bruisser les barbelés des maisons de banlieue.

Invisible mais partout présent,comme un influx nerveux,le Grand Ordinateur règle le ballet des marionnettes: un petit tour par ci,quelques frissons par là, n'oubliez pas la caisse à la sortie -TVA en sus- .

Ah ! le Printemps .

NOTION DE SYSTEME

Système logiciel et systeme machine

c. bianco

C.R. Subject classification informatics: C53 C54 D31 D41

Résumé.

Prenant en compte le fini-illimité dans le temps,dans le but de le séparer du fini-illimité dans l'espace pour les besoins de la démonstration,on se pose deux problèmes d'organisation du déroulement à l'intérieur de la mémoire centrale.L'un des problèmes correspond un peu à ce qu'on nommait jadis le batch-processing, et l'autre l'utilisation collective en temps partagé.L'étude de ces systèmes sous les aspects de codage et d'intégration permet de dégager les notions de système logiciel et de système machine et d'enrichir et compléter celle d'insertion.

NOTION DE SYSTEME

SYSTEME LOGICIEL,SYSTEME MACHINE.

L'ORDINATEUR FORMEL.

La machine formelle est en elle-même une sorte de processeur dont la description est l'algorithme de la machine formelle,et travaillant sur la mémoire centrale.J'ai négligé les instructions qui traitent des files,il va donc me falloir rajouter des mécanismes d'échange pour obtenir un véritable ordinateur.Je vais partir du plus simple,qui est manuel,pour obtenir une sorte d'ordinateur minimum ; et ultérieurement je remplacerai une partie des opérations manuelles par des manoeuvres automatiques,ce qui aura pour effet,chaque fois de créer un élément du système.

UN ORDINATEUR MINIMUM.

Au paragraphe de la "Structure du code de l'instruction" j'ai discuté du contenu de la case,et je me suis arrêté,pour l'exemple, à 13 bits.Je m'en tiendrai à cette taille,mais ce que je vais décrire pourrait s'appliquer à toute autre taille.

Je vais construire un tableau de bord qui va me permettre de contrôler le fonctionnement de mon ordinateur.Je vais le munir de divers visualisateurs destinés à montrer l'information manipu-

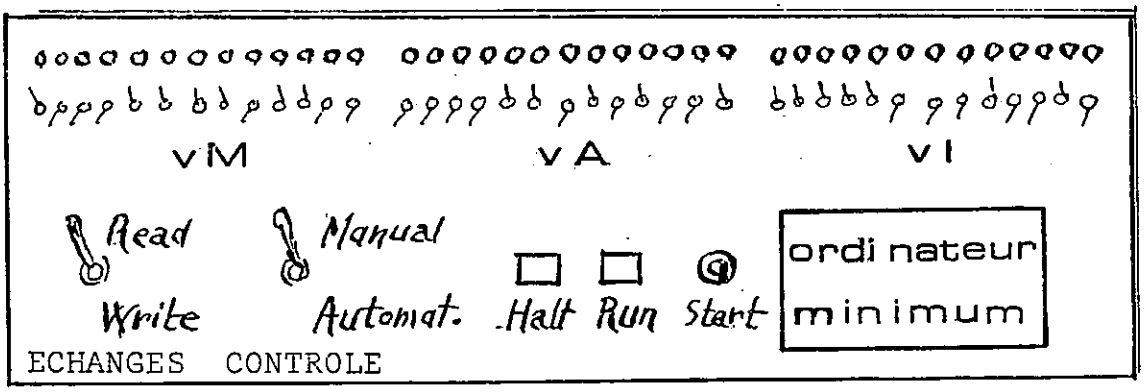
lée,et de diverses clefs dont les jeux de positions vont me servir à réaliser les échanges.

a) J'accroche au tableau de bord trois visualisateurs ainsi construits:
3 lignes de 13 néons qui peuvent être allumés ou éteints,en correspondance avec chacun de ces néons et juste en dessous,je place une clé à deux positions: haut et bas.Je grave sur la plaque le nom des trois visualisateurs: vM,vA et vI.

b) Je place deux grandes clefs de manoeuvre: ECHANGE et CONTROLE,chacune à deux positions.La clé ECHANGE peut se mettre en Read ou en Write.La clé CONTROLE en Manuel ou Automatique.

c) J'ajoute deux voyants lumineux: HALT et RUN,traditionnellement le premier est rouge l'autre est vert.

d) Enfin je place un bouton-poussoir: START.



Sur les néons de vA et vI sont visualisées les positions des clées correspondantes,de la façon suivante :

Clé en haut = néon allumé = 1

Clé en bas = néon éteint = 0

Sur les néons de vM sont visualisées soit les valeurs de la case mémoire sélectée,soit celles qui sont forcées par les clefs, selon que la clé ECHANGES est en Read ou en Write.

Les diverses fonctions représentées par les actions sur les clées ECHANGE , CONTROLE et START sont rassemblées dans le tableau ci-après:

Clef de CONTROLE	Automatique	Manuel	
Clef d' ECHANGE	Inefficace	Read	Write
ORDINATEUR	Pousser le bouton START charge en I ce qui est affiché en vI. Et lance la MF sur sa 1re ligne.	Sur vM est affiché le contenu de la case dont l'adresse est affichée en vA.	La valeur affichée en vM est envoyée à l'adresse affichée sur vA quand on presse START.
HALT	Eteint	Allumé	Allumé
RUN	Allumé	Eteint	Eteint

T5

Préparer un calcul consiste donc à décrire le code d'un programme - celui qu'on veut dérouler - avec son implantation en mémoire,à préparer l'implantation de la configuration sur laquelle il s'applique,à préparer également la configuration de la machine formelle et enfin calculer la valeur à fournir pour I.

Partant d'un exemple de programme,je vais montrer ce qu'est l'algorithme de mise en place en mémoire de toute l'information correspondante par la manipulation au tableau de bord. Auparavant,je ferai quelques remarques sur la validité des codes du programme.

Cas des erreurs.

Si en fin d'aiguillage (A1) on n'a pas reconnu de code opérateur,ou si on est parvenu en ERREUR 1,ERREUR 2 ou ERREUR 3,il serait bon de signaler que le programme ne signifie plus rien. En effet il peut s'agir d'une faute de codage,et à partir de là, ce qui suit est ininterprétable.On peut traiter ce cas ainsi:

On rajoute au tableau de bord un voyant blanc V relié à une case E par le jeu des valeurs suivantes.

E	V
0	éteint
1	allumé

et je rajoute dans l'algorithme de la machine formelle,en fin de A1,en ERREUR 1,en ERREUR 2 et en ERREUR 3 les instructions:

$[_E := 1$;

Ebloc : vers Ebloc ;

et je rajoute également au bouton START la propriété de réaliser dans l'ordre les deux opérations:

$[_E := 0$;

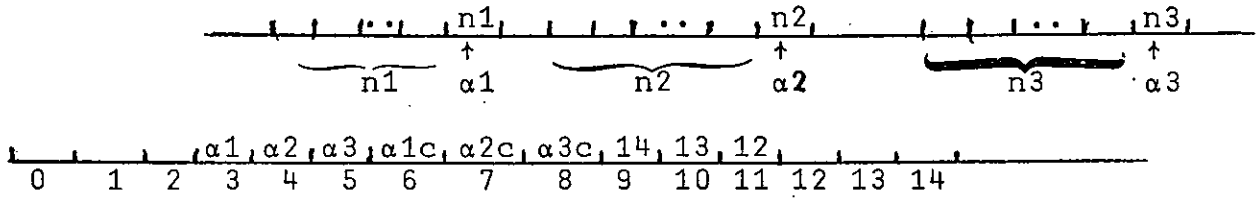
vers entrée ;

A titre d'exemple pour montrer comment utiliser l'ordinateur minimum,je construis un petit programme,que je code,ainsi que la configuration sur laquelle il s'applique.Je montre ensuite comment l'introduire et le lancer.

Addition de deux nombres à longueur quelconque.

Je prends le cas de nombres entiers trop grands pour tenir dans une case,et dont je suppose que la valeur est morcelée dans une suite de cases suffisamment longue.Je note donc,en mémoire et par adresses décroissantes: le nombre de cases occupées par

la valeur du nombre représenté,puis les morceaux du nombre,allant des poids faibles aux poids forts.L'adresse pour accéder à cette représentation du nombre est celle de la case qui contient la dimension de ce nombre.La représentation de la somme aura donc la dimension de la plus longue des données,éventuellement plus un.




```
        début
        param 3 ;
        index 3 ;
        var loc 3 : 1 , 1 , 1 ;
        [11,0 := 0 ;
        si [3,0 > [4,0 vers e1 ;
        [9,0 := [4,0 ;
        [10,0 := [3,0 ;
        [6 := [4 - 1 ;
        [7 := [3 - 1 ;
        [5,0 := [4,0 + 1 ;
        vers e2 ;
e1 :    [5,0 := [3,0 + 1 ;
        [9,0 := [3,0 ;
        [10,0 := [4,0 ;
        [6 := [3 - 1 ;
        [7 := [4 - 1 ;
e2 :    [8 := [5 - 1 ;
e4 :    si [10,0 = 0 vers ef ;
```

```
        [8,0 := [6,0 +k [7,0 ;
        si [ω = 1 vers e3 ;
        [8,0 := [8,0 +k [11,0 ;
        [11,0 := [ω ;
e5 :    [6 := [6 - 1 ;                  Addition tranche par
        [7 := [7 - 1 ;                  tranche.
        [8 := [8 - 1 ;
        [10,0 := [10,0 - 1 ;
        [9,0 := [9,0 - 1 ;
        vers e4 ;
e3 :    [8,0 := [8,0 +k [11,0 ;         Addition du report pré-
        [11,0 := 1 ;                    cédent,réservation du
        vers e5 ;                       report actuel.
ef :    si [9,0 = 0 vers eff ;          Propagation du report
        [8,0 := [6,0 +k [11,0 ;         quand le nombre le plus
        [11,0 := [ω ;                   court est épuisé.
        [8 := [8 - 1 ;
        [6 := [6 - 1 ;
        [9,0 := [9,0 - 1 ;
        vers ef ;
eff :   si [11,0 ≠ 0 vers eff1 ;        Correction de la lon-
        [5,0 := [5,0 - 1 ;              gueur du résultat s'il
        vers fin ;                      n'y a pas débordement.
eff1 :  [8,0 := [11,0 ;
fin :   fin
```

J'insiste sur le fait que dans cet ordinateur minimum,c'est toutes les files externes qui sont remplacées par des opérations manuelles tout aussi illimitées.

Codage,chargement et déroulement.

Pour charger le programme en mémoire,il me faut au préalable le coder.En même temps que je construis le code,je lui affecte une localisation.Ensuite je définis les diverses configurations ainsi que leur implantation.J'ai besoin de la configuration de ce programme,que je place en adresse 50 par exemple,puis il faut les données du problème,c'est-à-dire les paramètres effectifs du programme,je les place en case 100 et suivantes,enfin il faut la configuration de la MF,que je place en adresse 0.

Le code du programme,je le charge à partir de 200.La valeur de I au départ est de 0 car elle désigne la configuration de la MF.Et,respectivement dans chacune des configurations je porte comme paramètres effectifs les distances par rapport à chaque origine.

Etiq.	N°	opeur	opérandes					
	200	0						
	201	15	3	3	3	1	1	1
	208	6,1,4	11	0	0			
	212	9,1,1	3	0	4	0	32	e1
	218	6,1,1	9	0	4	0		
	223	6,1,1	9	0	3	0		
	228	3,0,0,4	6	4	1			
	232	3,0,0,4	7	3	1			
	236	2,1,1,4	5	0	4	0	1	
	242	13	26	e2				
e1	244	2,1,1,4	5	0	3	0	1	
	250	6,1,1	9	0	3	0		
	255	6,1,1	10	0	4	0		
	260	3,0,0,4	6	3	1			
	264	3,0,0,4	7	4	1			
e2	268	3,0,0,4	8	5	1			
e4	272	7,1,4	10	0	0	64	ef	
	277	2,1,1,1	8	0	6	0	7	0
	284	7,2,4	1	39	e3			
	287	2,1,1,1	8	0	8	0	11	0
	294	6,1,2	11	0				
e5	297	3,0,0,4	6	6	1			
	301	3,0,0,4	7	7	1			
	305	3,0,0,4	8	8	1			
	309	3,1,1,4	10	0	10	0	1	

	315	3,1,1,4	9	0	9	0	1	
	321	13	-49	e4				
e3	323	2,1,1,1	8	0	8	0	11	0
	330	6,1,4	11	0	1			
	334	13	-37	e5				
ef	336	7,1,4	9	0	0	31	eff	
	341	2,1,1,1	8	0	6	0	11	0
	348	6,1,2	11	0				
	351	3,0,0,4	8	8	1			
	355	3,0,0,4	6	6	1			
	359	3,1,1,4	9	0	9	0	1	
	365	13	-29	ef				
eff	367	8,1,4	11	0	0	13	ef1	
	372	3,1,1,4	5	0	5	0	1	
	378	13	7	fin				
eff1	380	6,1,1	8	0	11	0		
fin	385	1						
Etiq.	N°	op.eur	opérandes					

Configuration de données.

N°	valeurs						
100	4	5	6	3			
104	9	3	8	7	4		
111	0	0	0	0	0	0	0

Configuration du programme.

N°	valeurs						
50	0	0	0				
53	50	54	60				

Configuration de la MF.

0	0	0	0				
3	200	50					

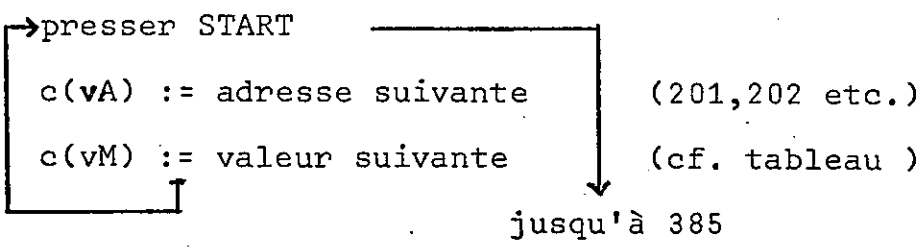
La mise en place en mémoire de cette information se réalise au moyen de l'algorithme manuel suivant:

CONTROLE : Manual

ECHANGES : Write

c(vA) := 200 (Positionner 200 aux clés vA)

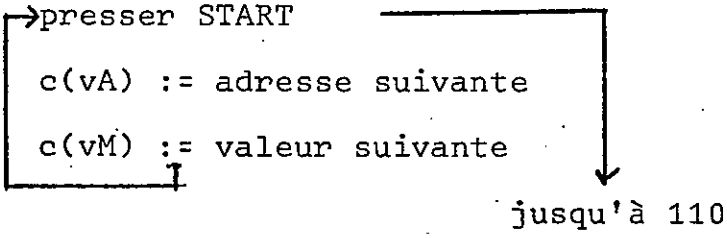
c(vM) := 0 (Positionner 0 aux clés vM)



Puis ensuite:

c(vA) := 100

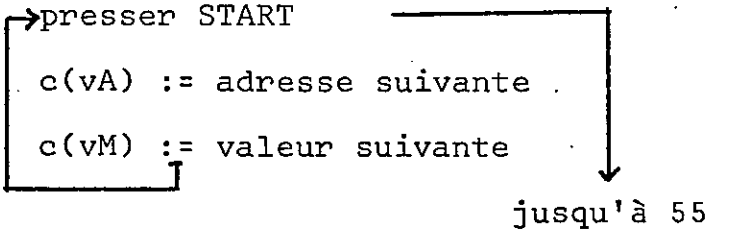
c(vM) := 4



Puis encore:

c(vA) := 50

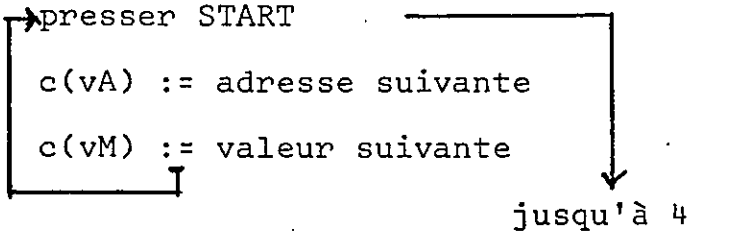
c(vM) := 0



Enfin:

c(vA) := 0

c(vM) := 0



Et enfin:

c(vI) := 0 (Positionner 0 aux clés de vI)

presser START Ce qui lance le calcul.

SYSTEMES ELEMENTAIRES

TRAITEMENT DE L'ILLIMITE.

Je vais essayer d'automatiser une partie de "l'illimité dans le temps".Pour résoudre ce problème j'avance deux méthodes dont on verra le rapport qu'elles peuvent comporter avec les réalisations pratiques.L'intérêt des deux exemples que je vais décrire complètement,réside dans le fait qu'ils ne matérialisent qu'une seule propriété à la fois.Cela permet de raisonner plus facilement sur la notion à mettre en oeuvre,en l'occurence la notion de l'insertion,et d'en étudier l'impact,dégagé d'influences parasites.

Je distingue le "système vertical" qui enchaîne le déroulement d'un ensemble de programmes de façon qu'ils soient traités tous et complètement,l'un après l'autre.

Et puis le "système horizontal" qui,lui,déroule un ensemble de programmes en parallèle.Par permutation il déroule un élément de chacun,c'est-à-dire qu'il déroule un élément d'un programme, puis passe au programme suivant dont il déroule un élément,puis encore au suivant jusqu'à ce que tous les programmes soient traités.

Dans les deux cas je suppose les programmes à dérouler préintroduits en mémoire.le problème de l'approvisionnement de la mémoire centrale ou "fini illimité dans l'espace",est traité à

part.

J'ajoute une précision terminologique: dans ce qui suit j'emploierai les termes "adresse" et "accès" pratiquement avec le même sens,celui d'une distance en nombre de cases par rapport à l'origine de la configuration du programme décrit.

SYSTEME VERTICAL.

Je fais donc l'hypothèse qu'au moment où le système en question commence à se dérouler,se trouve déja en place dans la mémoire centrale la suite des codes des programmes à traiter.Je désigne par: p1,p2, ... pf les adresses du début de ces codes par rapport à l'origine de la configuration du système.Je fais également l'hypothèse qu'une part de la mémoire a été retenue pour chacune des configurations de travail de ces programmes.Et j'appelle d1,d2, ... df les adresses du début de ces configurations rapportées à la même origine.

Chacun des programmes pouvant être constitué de plusieurs procédures,il faut prévoir des insertions au déroulement.cela signifie que chaque part doit être assez large pour contenir l'étalement maximum des configurations dû aux insertions.

On constate au passage que se posent plusieurs problèmes connexes qui doivent être résolus dans un système général:

1) Contrôle des implantations en mémoire des codes-programmes et des parts réservées aux configurations.

2) Contrôle du débordement possible de chaque configuration de la part qui lui est réservée.

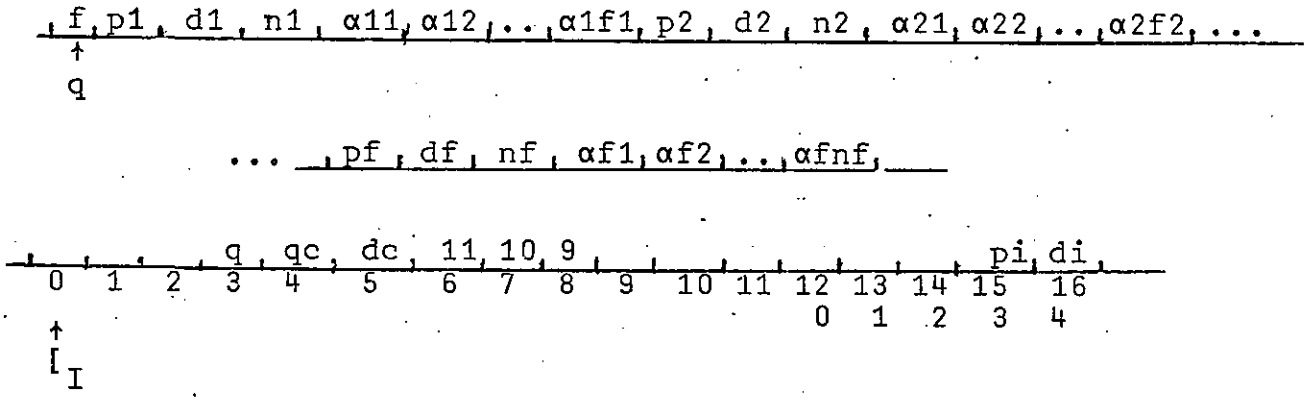
Je me contente de souligner l'existence de ces questions qui sont soulevées dans d'autres chapitres.

SYSTEME VERTICAL INTEGRE.

Je construis une première version dans laquelle le système est une sorte d'extension de la MF.Le système prépare la configuration du programme à dérouler:

Il met en place les paramètres effectifs,puis il se renvoit par un simple "aller à " à la MF.Cette dernière,après avoir achevé le déroulement du programme,retourne au système également avec un aller à.

Les configurations se présentent ainsi:



La machine formelle MF telle que je l'ai définie dans le bulletin N°9,ne possède pas le moyen de détecter quand un programme s'achève,elle ne sait pas reconnaître le fin qui correspond au premier début qu'on lui a soumis.Il faut donc la munir de cette propriété supplémentaire.

Ce qui différencie ce premier couple début,fin des autres, c'est la première insertion qui est réalisée artificiellement par le système.On peut adopter la solution suivante:

Constatant qu'aucune adresse de retour systématique (Contenu de la case 2 de la configuration) ne peut être nulle,par construction,il suffit donc que le système injecte un zéro dans cette case 2 de la première configuration.La MF qui rencontre un fin commence par tester si l'adresse de retour est différente

de zéro,auquel cas elle calcule comme précédemment,sinon elle se renvoit dans le système à la ligne de programme qui suit celle qui a commuté vers MF.

```
efin : si [4,2 = 0 vers retsys ;
       [3 := [4,2 ;
       [4 := [4 - [4,1 ;
       vers entrée ;
```

Alors l'ensemble système vertical - machine formelle se construit ainsi:

```
       début
       par 1 ;
       index 2 ;
       vl 3 : 1 , 1 , 1 ;
       [6,0 := [3,0 ;
       [4 := [3 + 1 ;
sys3 : si [6,0 = 0 vers sysf1 ;
       [15 := [4,0 - 12 ;
       [16 := [4,1 - 12 ;
       [8,0 := [4,1 ;
       [5 := [4,1 + 3 ;
       [4 := [4 + 2 ;
       [7,0 := [4,0 ;
       [4 := [4 + 1 ;
sys2 : si [7,0 = 0 vers sys1 ;
       [5,0 := [4,0 - [8,0 ;
       [7,0 := [7,0 - 1 ;
       [5 := [5 + 1 ;
       [4 := [4 + 1 ;
```

pi : accès au code-programme.

di : accès à la configuration.

ni : nombre de paramètres.

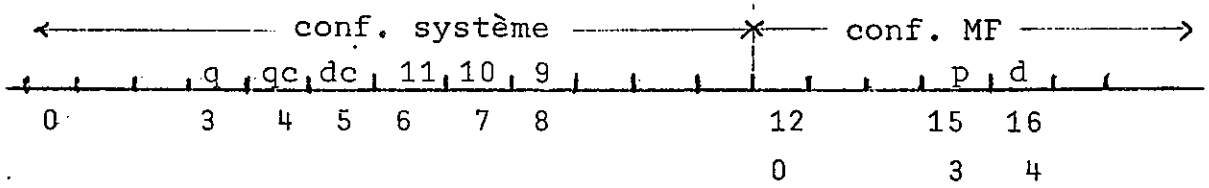
αij : accès à la valeur du paramètre / origine de la configuration système.

f : nombre de programmes à dérouler.

Transmission des paramètres du programme à dérouler.

```
        vers sys2 ;
sys1 : [I :=[I + 12 ;          Passage à la configuration
        vers MF ;              de MF.
retsys : [I := [I - 12 ;       Retour à la configuration du
        [6,0 := [6,0 - 1 ;     système.
        vers sys3 ;
sysf1 : vers sysf1 ;
```

La structure des configurations sur lesquelles s'appliquent ces deux programmes est la suivante:



En décrivant ainsi mon système vertical,je m'apperçois qu'en fait,je réalise une sorte d'insertion en créant la configuration du programme à dérouler,puis en y plaçant les paramètres effectifs.Force m'est de constater que l'insertion déja traitée par la MF,ne peut convenir car,au moment où je dois écrire cette insertion,je ne peux connaître le nom du programme à insérer qui est par définition changeant d'un traitement à l'autre.Je crée donc une insertion spécifique: l'insertion paramétrique .

Insertion paramétrique.

L'instruction qui réalise l'insertion paramétrique,comporte un seul paramètre dans sa représentation,et ce paramètre -un index- permet d'accéder à une liste contenant:

a) L'accès au code programme à insérer.

b) L'accès à la configuration qui correspond à ce programme.

c) Le nombre de paramètres effectifs.

d) La liste des paramètres effectifs.

Il me faut rajouter à la MF le traitement de cette instruction supplémentaire.Je lui choisis donc un codage qui va tenir en deux cases:

1) Le code "inspar".

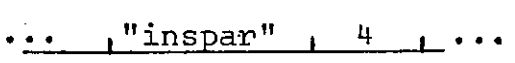
2) Le numéro de l'index.

Exemple

L'instruction:

inspar 4 ;

aura pour code en mémoire:



Traitement de l'instruction paramétrique en MF.

Je rajoute une instruction dans l'algorithme A1,qui décrit l'aiguillage des codes instructions:

si $[_{6,0}$ = "inspar" vers eip ;

Et je rajoute également le traitement de l'instruction elle-même:

```
eip  : [10 := [3,1 + [4 ;
       [10 := [10,0 + [4 ;
       [11 := [10 ;
       [12 := [10,2 + 2 ;
       [12 := [12 + [4 ;
       [12,0 := 0 ;
       [6,0 := [10,3 ;
       [10 := [10 + 4 ;
eip1 : si [6,0 = 0 vers eipf ;
```

Calcul des accès aux paramètres effectifs pour la configuration du programme à insérer ,à partir de celle du programme insérant.

```
        [12,1 := [10,0 - [11,2 ;
        [12 := [12 + 1 ;          Mise en place des paramètres
        [10 := [10 + 1 ;          effectifs dans la configura-
        [6,0 := [6,0 - 1 ;        tion du programme à insérer.
        vers eip1 ;
eipf :  [3 := [11,1 + [4 ;
        vers entrée ;
```

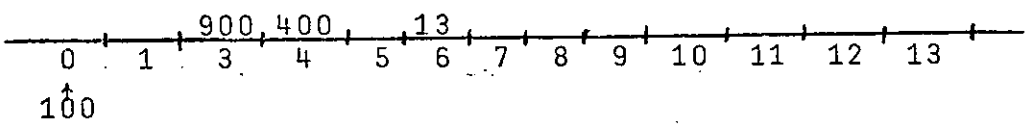
Je rappelle les hypothèses de constitution des diverses configurations employées dans ce problème.Je dispose d'un programme insérant (ici le système vertical paramétrique) et d'un programme à insérer,l'un de ceux dont l'adresse est fournie par la file d'adresse q,définie plus haut.Par choix j'ai décidé que toutes les adresses relatives à cette file sont calculées par rapport à l'origine de la configuration du programme insérant,que ce soit: q , les pi, les di,les αjk.Dans l'exemple qui suit je place cette file des programmes à dérouler en adresse: q = 200 .

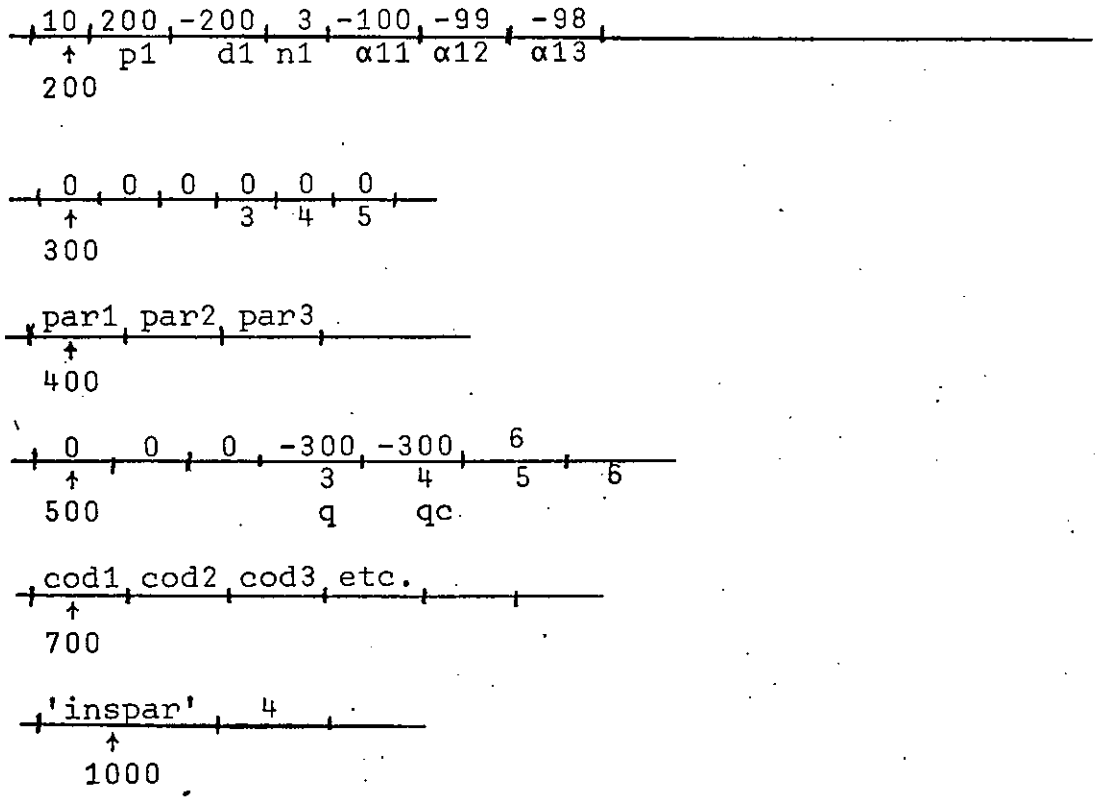
En adresse 100 je place la configuration de la MF.Si je désigne par p et d,les accès au code et à la configuration du programme insérant,j'ai p = 900 et d = 400 si code et configuration sont localisés en 1000 et 500 .

Je m'intéresse par exemple au premier programme à dérouler, dont les accès p1 et d1 sont localisés en conséquence en 201 et 202 .Je place les zones correspondantes en 700 et 300 .

La configuration du programme insérant étant située en 500 et p1 et d1 étant calculés par rapport à cette adresse:

p1 = 200 et d1 = -200





Le calcul du traitement de cette insertion consiste à placer un zéro en 302 pour le retour après déroulement complet du programme,et à placer: 100 , 101 , 102 respectivement en 303 , 304 , 305 accès aux valeurs des paramètres effectifs par rapport à l'origine de la configuration de ce programme.

SYSTEME VERTICAL PARAMETRIQUE.

Le système vertical devient alors très simple quand on l'applique sur une liste de programmes dont on vient de voir un exemple.Son rôle se borne à contrôler le nombre de programmes à dérouler,et à calculer un index qui fournisse dans la file des programmes l'accès aux caractéristiques de celui qu'il va falloir insérer.

```
Systpar : début
        par 1 ;
        index 1 ;
        vl 1 : 1 ;
        [5,0 := [3,0 ;
        [4 := [3 + 1 ;
sys1 : si [5,0 = 0 vers sysf1 ;
        inspar 4 ;
        [4 := [4 + [4,2 ;
        [4 := [4 + 3 ;
        [5,0 := [5,0 - 1 ;
        vers sys1 ;
```

Concernant ces deux versions du même système,je peux d'ores et déja faire une remarque dont l'importance est relative à leur aspect linguistique.

Le premier,système vertical intégré,est de même niveau que la MF,les deux constituent un même algorithme,qui se place au méta-niveau par rapport aux programmes à dérouler.

L'autre,système vertical paramétrique,est de même niveau que les programmes à dérouler,et c'est l'insertion paramétrique qui les met au même niveau.Dans ce cas,seule la MF est de méta-niveau,elle comporte cependant en elle-même un système minimum. En effet on utilise la MF telle qu'elle est définie au N°9 du Bulletin,et,par construction,on constate qu'elle boucle perpétuellement sur elle-même.

SYSTEME HORIZONTAL.

Comme dans le cas du système vertical,je suppose en place un jeu de "f" programmes,avec f quelconque pourvu que tout tienne en mémoire.Mais je vais faire dérouler tous ces programmes en parallèle.En fait je fais dérouler un petit morceau de chacun par rotation.Pour réaliser ce découpage des tâches en vue d'une distribution du travail,soit on travaille au chronomètre en lançant un programme et en l'interrompant de force au bout d'un temps déterminé,soit on structure le déroulement des programmes de façon qu'ils retournent eux-même au distributeur de tâches.Et là,je définis une notion de structuration: l'autojection,et une nouvelle notion d'insertion: l'insertion fractionnée.

SYSTEME HORIZONTAL INTEGRE.

Pour la démonstration je définis un quantum de travail honnêtement minimum: le déroulement d'une instruction des programmes à dérouler.Ceci n'est pas véritablement un lit de Procuste,car on constatera que rien au fond n'empêche de choisir des quanta différents.En fait cette notion prend tout son intérêt à propos des langages évolués: le quantum minimum peut alors être choisi pour correspondre à la sémantique d'une phrase élémentaire du langage dans la mesure où on sait qu'elle est finie-bornée.

Le rôle du système consiste à demander à la MF de prendre en compte le code de l'instruction en cours,de préparer le calcul de la suivante,et de retourner au système qui alors demande à la MF le même service pour le programme suivant.

Il est donc nécessaire que la MF puisse disposer d'autant de

configurations de travail qu'il y a de programmes,pour conserver l'information du déroulement d'une instruction sur celui de la suivante.Entre ces deux moments la MF calcule sur des programmes différents.

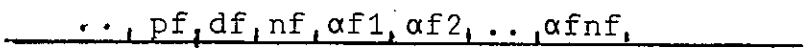
A l'initialisation le système doit donc créer ces configurations.Ensuite,pendant le déroulement des programmes,il passe de l'une à l'autre avant de se renvoyer à la MF.

```
systHI : début
         par 1 ;
         ind 4 ;
         vl 2 : 1 , 1 ;
         [8,0 := [3,0 ;
         [5 := 9 ;                    Préparation de la file des
         [7 := [3,0 * 2 ;             accès aux configurations de
         [7 := [7 + [5 ;              la MF,à partir de 9 jusqu'à
         [6 := [5 ;                  (2 * f) + 9 .Et des configu-
         [4 := [3 + 1 ;               rations à partir de (2*f)+9.
sys2 : si [8,0 = 0 vers syst1 ;
         [6,0 := [7 ;
         [7,3 := [4,0 - [6,0 ;
         [7,4 := [4,1 - [6,0 ;        Préparation du contenu des
         [6,1 := "deb" ;              configurations de MF.
         [7,5 := [6 - [6,0 ;
         [7,5 := [7,5 + 1 ;
         [7,6 := [4 - [6,0 ;
         [7,6 := [7,6 + 2 ;
         [7 := [7 + 18 ;
         [6 := [6 + 2 ;
```

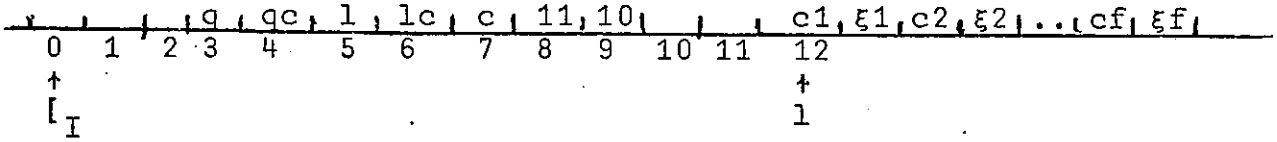
```
        [4 := [4 + [4,2 ;
        [4 := [4 + 3 ;
        [8,0 := [8,0 - 1 ;
        verssys2 ;
syst1 : [6 := [5 ;                    Déroulement des programmes
        [8,0 := [3,0 ;                en parallèle,contrôle de
sys4 : si [8,0 = 0 vers syst1 ;       fin de programmes et saut
        [7 := [6,0 ;                  des programmes achevés.
        si [6,1 = "terminé" vers sysf ;
        si [6,1 = "arrêt" vers sys5 ;
        [7,1 := [I ;
        [I := [I + [6,0 ;
        vers MF ;
retsys : [I := [1 ;
sys5 : [8,0 := [8,0 - 1 ;
        [6 := [6 + 2 ;
        vers sys4 ;
sysf : [9,0 := [9,0 + 1 ;
        si [9,0 = [3,0 vers fin ;
        [6,1 := "arrêt" ;
        vers sys5 ;
fin :   vers fin ;
        fin
```

Pour illustrer le domaine d'applications d'un tel système,je construis l'image suivante des configurations:

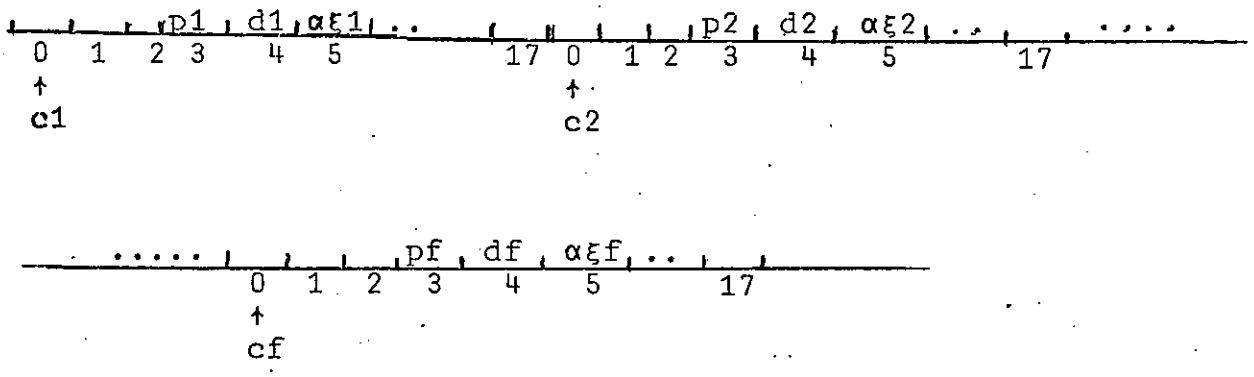
```
, f , p1, d1, n1, α11, α12, .. ,α1n1, p2, d2, n2, α21, α22, .. ,α2n2, ....
  ↑
  q
```



Ceci montre la structure de la file des programmes à dérouler, voici alors la configuration du système,suivie par la file des couples ci,ξi ,dans laquelle ci est l'accès à la configuration de MF quand celle-ci traite du programme N°i,et ξi est l'état du traitement de ce même programme,tel qu'il est défini ci-dessous.



Enfin la suite des configurations de MF:



La variable ξi peut prendre les quatre valeurs ainsi détaillées:

1) "début" : mis en place par le système avant de lancer le déroulement du programme correspondant.

2) "en cours" : quand la MF rencontre "début" elle le remplace par "en cours",cela lui permet d'initialiser son calcul.

3) "terminé" : valeur qui remplace "en cours" quand la MF

rencontre le dernier fin du programme qui se déroule.

4) "arrêt" : quand le système rencontre "terminé" il est informé de la fin du traitement du programme correspondant,il le clôt et note "arrêt" pour ne plus s'attarder sur ce programme en continuant à distribuer ses quanta.

Modifications de la MF.

Ce nouveau jeu que représente le système horizontal intégré exige quelques modifications de la MF.Le traitement de l'opérateur fin devient :

```
efin : si [4,2 = 0 vers efret ;
       [3 := [4,2 ;
       [4 := [4 - [4,1 ;
       vers retsys ;
efret : [5,0 := "terminé" ;
       vers retsys ;
```

Il faut remplacer également tous les "vers entrée ;" par des "vers retsys ;" .

SYSTEME HORIZONTAL FRACTIONNE.

Il est possible d'imaginer une instruction d'insertion qu'on dira "fractionnée" et qui permet au système d'insérer un programme de son choix dont il ne se déroulera qu'un élément d'algorithme fini-borné,par exemple une seule instruction.

A titre d'initialisation la nouvelle version de système -le système horizontal fractionné- crée une file qui est destinée

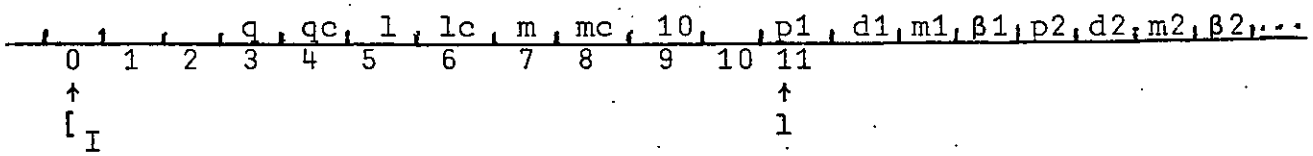
à la machine formelle,et qui contient pour chaque programme à dérouler quatre paramètres caractéristique.Il s'agit des accès au code du programme,à sa configuration,à sa variable d'état,et à sa liste des paramètres effectifs.

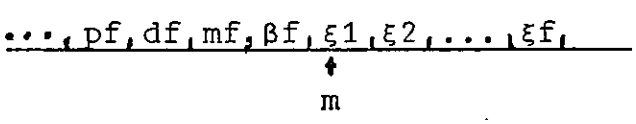
Une deuxième boucle du système contrôle le déroulement des programmes à l'aide d'une insertion fractionnée.

```
sysHFR : début
         par 1 ;
         ind 5 ;
         vl 1 : 1 ;
         [4 := [3 + 1 ;
         [5 := 11 ;
         [6 := [5 ;
         [8 := [3,0 * 4 ;
         [8 := [8 + [5 ;
         [7 := [8 ;
sys0 :   si [6 = [7 vers suit ;
         [6,0 := [4,0 ;          Constitution de la file des
         [6,1 := [4,1 ;          paramètres de chaque pro-
         [6,2 := [8 ;            gramme: pi,di,αξi,βi .
         [6,3 := [4 + 2 ;
         [6 := [6 + 4 ;
         [4 := [4 + [4,2 ;
         [4 := [4 + 3 ;
         [8,0 := "début" ;       Initialisation de l'état du
         [8 := [8 + 1 ;          déroulement.
         vers sys0 ;
suit :   [9,0 := 0 ;
```

```
sysf : [6 := [5 ;
       [8 := [7 ;
syst1 : si [6 = [7 vers sysf ;          Boucle de déroulement
       si [8,0 = "arrêt" vers sys4 ;    des programmes.
       insfract (6) ;
       si [8,0 = "terminé" vers sys2 ;
sys4 : [6 := [6 + 4 ;
       [8 := [8 + 1 ;
       vers syst1;
sys2 : [9,0 := [9,0 + 1 ;
       si [9,0 = [3,0 vers sysfin ;
       [8,0 := "arrêt" ;
       vers sys4 ;
sysfin : vers sysfin ;
       fin
```

La file des programmes est de même structure que dans les exemples précédants,je ne la répète pas,mais la configuration du système a la forme suivante:





Il faut maintenant montrer que le traitement d'une telle insertion fractionnée est possible,reprenons la machine formelle.

Machine formelle et insertion fractionnée.

L'insertion fractionnée localisée dans une instruction,ramène le système au niveau linguistique des programmes à dérouler.Il faut montrer ce que devient une MF capable de traiter une telle insertion.Dans ce but j'en définis la syntaxe et la sémantique.

Le code de l'instruction d'insertion fractionnée comporte un code opération "insfract" et un indicateur d'index qui est un numéro.L'index ainsi désigné appartient à la configuration du programme qui contient l'insertion (par exemple: le système) et il permet d'atteindre à une liste de quatre paramètres décrite ci-dessus:

pi,di,mi,βi

pi est l'accès au code du programme à dérouler.

di est l'accès à sa configuration.

mi est l'accès à la variable d'état du déroulement.

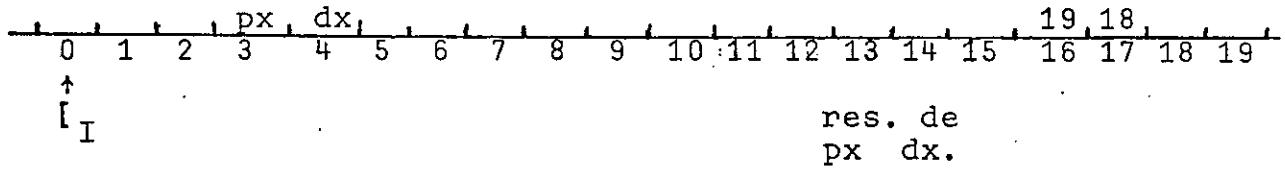
βi est l'accès à la liste des paramètres effectifs.

La variable d'état prend les mêmes valeurs que dans le cas précédent: "début" , "en cours" , "terminé" , "arrêt" .

Quand la MF rencontre une insertion fractionnée avec "début" en état de déroulement,elle réalise la mise en placé des paramètres effectifs dans la configuration dont l'accès est di,à partir de la liste dont l'accès est βi .Et elle fait passer l'état à "en cours".En même temps elle met à zéro la case 2 de cette configuration qui est celle du programme à lancer,en vue d'en préparer l'arrêt,et ceci comme pour l'exemple précédent.

A la rencontre d'un fin ,avant de calculer le retour systématique,la MF vérifie la valeur de l'adresse de retour.S'il y a zéro cela signifie qu'on est tombé sur le dernier fin ,dernier au sens du déroulement.Il suffit alors de faire passer l'état du

déroulement à "terminé".



La configuration de cette nouvelle MF doit comporter quelques cases supplémentaires.Il en faut deux pour réserver px et dx de la procédure insérante: ce sont les cases 13 et 14.Pour accéder à l'état du déroulement on utilise la case 15,et pour contrôler le transfert des paramètres effectifs: la case 17.La case 16 sert à différencier après traitement les instructions du système de celles du programme à dérouler.

```
MF :    début
        par 2 ;
        ind 11 ;
        vl 2 : 1 , 1 ;
        [16,0 := 0 ;
entrée : [6,0 := [3,0 ÷ 2^9 ;
        si [6,0 = "début" vers edeb ;
        ---   ---   ---   ---   ---
        si [6,0 = "insfract" vers ifrac ;
        ---   ---   ---   ---   ---
ifrac : [16,0 := 1 ;
        [10 := [4 + [3,1 ;
        [10 := [10,0 + [4 ;
        [15 := [10,2 + [4 ;
        si [15,0 = "début" vers ifrdeb ;
ifrf :  [13 := [3 ;
```

```
        [14 := [4 ;                    Réservation de px et dx en
        [3 := [10,0 ;                  13 et 14 ,et appel de pi,di
        [4 := [10,1 ;                  en 3 et 4 .
        vers entrée ;
ifrdeb : [11 := [4 +[10,3 ;            Initialisation du calcul
        [12 := [4 + [10,1 ;            pour un programme dont on
        [12,0 := 0 ;                   rencontre la première ins-
        [15,0 := "en cours" ;          truction.
        [13 := [11 + 1 ;
        [17,0 := [11,0 ;
        [14 := [10,1 ;
ifr1 : si [17,0 = 0 vers ifrf ;
        [12,3 := [13,0 - [10,1 ;       Paramètre effectif recalcu-
        [12 := [12 + 1 ;               lé pour l'origine dont l'ac-
        [13 := [13 + 1 ;               cès est: di.
        [17,0 := [17,0 - 1 ;
        vers ifr1 ;
```

Je ne donnerai qu'un seul autre exemple de traitement d'opérateur,celui de fin.Les autres ne présentent aucune difficulté.

```
efin : si [4,2 = 0 vers efdef ;
        [3 := [4,2 ;
        [4 := [4 - [4,1 ;
efin1 : si [16,0 ≠ 1 vers entrée ;
        [3 := [13 + 2 ;                Une telle séquence doit
        [4 := [14 ;                    être ajoutée à la fin de
        [16,0 := 0 ;                   chacun des traitements d'
        vers entrée ;                  opérateurs.
```

```
efdef : [15,0 := "terminé" ;
        vers efin1 ;
```

Là encore la MF qui demeure au méta-niveau comporte un système minimum,aussi au méta-niveau pour le système horizontal fractionné.

En outre,ce dernier type de système ne peut communiquer directement avec la MF,car celle-ci est de méta-niveau par rapport à lui.Il ne peut donc lui fournir directement une configuration pour chaque programme,comme dans le cas du système horizontal intégré. Il peut,par contre,créer une suite de jeux de paramètres ayant même signification (pi,di,mi,βi),mais auxquels la MF accède indirectement.C'est à cause de cela que la configuration de la MF se complique dans ce cas.On remarque de plus,que cette dernière MF,tant qu'elle ne rencontre pas d'instructions "insfract" fonctionne exactement comme celle décrite dans le N°9 du bulletin.

à suivre.

COMPILATION DE LA PROCEDURE FORMELLE SYMBOLIQUE

el ghazi el houssaini s.

C.R. Subject classification informatics: D34 B14

RESUME

Cet article donne une solution à la représentation sur une file support,des files du P.F.S. et une solution à leur débordement .

INTRODUCTION

Le but de la création du P.F.S. est d'introduire des notions supplémentaires par rapport à la procédure formelle,tout en restant trés proche des langages des procésseurs.

Ce qui a été apporté de nouveau,c'est à la fois les notions de statut et d'autojéction, et la commodité de désigner des éléments à l'aide des identificateurs dans une écriture de langage machine.

La P.F.S. est déstinée à décrire des compilateurs et systèmes autojéctifs,on se munit de quatre structures:

- Les files tableaux à arrangement répétitif;
- Les files dynamiques à arrangement disparâte;
- Les files messages à accès séquentiel;
- les files supports à structure assez souple pour supporter les files ci-dessus.

Les trois premières structures sont déstinées à prendre en compte les notions du fini borné; la file support et les extensions des files couvrent le champ du fini illimité.

FILE SUPPORT ET CONFIGURATION DES DONNEES

FILE SUPPORT

Travaillant sur une file support, je dois faire les hypothèses sur la manière dont elle est construite et les moyens dont je dispose pour la manipuler.

Une ligne de la file support étant définie par une valeur et un statut, je me munis en conséquence de deux tables: La table des valeurs et la table des statuts qui contiendront respectivement les valeurs et les statuts des lignes.

Pour ordonner ces lignes, je me donne une troisième table: la table des noms; à chaque ligne de la file support, je lui associe un élément de cette table composé de deux adresses: la première repère la valeur de la ligne dans la table des valeurs, la deuxiéme repère son statut dans la table des statuts.

Autres que les instructions de calcul sur les références et d'accès aux éléments de la file support, je me donne deux instructions supplémentaires: LIGNE et STATUT, la première permet de définir le support des valeurs d'une ligne et de noter son adresse dans la table des noms, la deuxième permet de définir le statut d'une ligne et de noter son adresse dans la table des noms.

EXEMPLE (cf: FIGURE1)

```
     support  1000,10,X(1):Y1,Y2;
              X:=12
              Y1:=Initial
I1            LIGNE Y1,X
I2            Y1 := STATUT (3),(4),(3),(2)
              Y2 :=5 ----> Y1
I3            LIGNE Y2,Y1
```

L'instruction I1 a pour effet de réserver dans "la table des valeurs" (là où il y a de la place) une zone de douze éléments minimums et de noter son adresse dans le premier composant du premier élément de "la table des noms".

L'instruction I2 a pour effet de mettre le statut dans "la table des statuts" et de noter son adresse dans le deuxième composant du premier élément de "la table des noms".

L'instruction I3 a pour effet de recopier dans le sixième élément de la table des noms les deux adresses du premier. Cette instruction permet d'avoir un support commun pour les valeurs de la ligne un et de la ligne six .

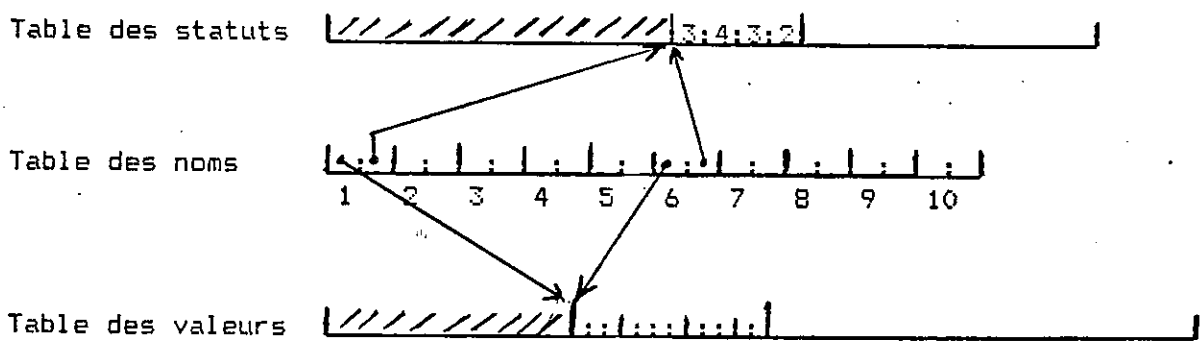


FIGURE 1 :SCHEMA D'UNE FILE SUPPORT.

On remarque, que le support des valeurs d'une ligne n'existera que si on effectue une instruction LIGNE.

Dans la suite, on représentera une file support par sa table des noms, sachant que chaque élément de cette table identifie une ligne, en repérant sa valeur et son statut s'ils existent .

CONFIGURATION DES DONNEES

On la matérialise en P.F.S., par une suite de lignes d'une file support(Cf.FIGURE 2).

On numérote les lignes de la configurations à partir de zéro, ces numéros sont pris par rapport à une origine qui se trouve dans une référence "SI" attachée à la file support. A chaque variable du programme image, on fait correspondre une ligne de la configuration (Le statut de la ligne étant déterminé par la nature de la variable) et on occupe les lignes dans l'ordre où apparaissent les variables dans le texte; on réserve toute fois les lignes de numéro zéro et un au débordement et à l'insértion.

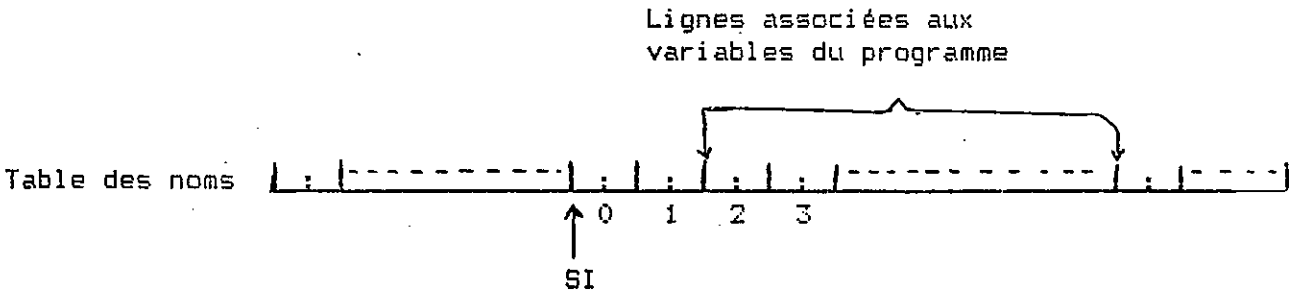


FIGURE 2 : SCHEMA DE LA CONFIGURATION DES DONNEES

LA FILE TABLEAU

DECLARATION DU TABLEAU

tableau NBLI (el1(e1),el2(e2),....,eln(en)) NBEX EXT NBLE:Ri,..,Rp

Où :

NBLI :Entier,
désigne le nombre initial des lignes du tableau.

eli(ei) :Identificateur,
désigne le ieme élément d'une ligne tableau,
représenté sur ei éléments-minimums .

NBEX :Entier,
désigne le nombre d'extensions possibles du tableau.

NBLE :Entier,
désigne le nombre de lignes par lesquelles on allonge le tableau ,en cas d'extension .

R1,..Rp :Identificateurs,
désignent les références attachées au tableau.

Tout ceci signifie,qu'on dispose au départ d'un tableau de "NBLI" lignes,dont chacune est composée de "n" éléments et qu'on peut étendre éventuellement ce tableau de "NBEX" fois "NBLE" lignes .

UTILISATION DU TABLEAU

Ri := Initial

Ri := Ultime

Ri := exp ----> Rj

Ri := exp <---- Rj

[Ri

[Ri,elj

Où "exp" est une expréssion entiére
i et j sont compris entre 1 et p.

LE COMPILATEUR GENERANT

On va décrire le travail effectué par le générateur,à la lecture de chacune des phrases qui s'appliquent sur un tableau.

DECLARATION DU TABLEAU

Tableau NBLI (el1(e1),el2(e2),...,eln(en)) NBEX EXT NBEX :R1,R2,...,Rp

Dans le programme image, on fait correspondre au tableau, trois sortes de variables locales:

- -Une variable indicée de "NBLI" lignes, qu'on appellera "variable initiale",qui existe dans la configuration par son nom et ses valeurs;
- -"NBEX" variables indicées de "NBLE" lignes chacune, qu'on appellera "variables d'extension", qui n' existent dans la configuration que par leurs noms ; à chaque extension du tableau, on créera le "support des valeurs" d'une variable d'extension;
- -"p" variables simples qui existent dans la configuration par leurs noms et leurs valeurs.

soit "N" le numéro de la première ligne libre dans la configuration des données; à la lecture de la phrase de déclaration le générateur fait correspondre (Cf.FIGURE 3):

-La ligne "N" avec un statut de (6(2),"NBLI"(e1,e2,...,en)) (*) à la variable initiale ;

-Réserve les lignes de (N +i) à (N + NBEX) aux variables d'extension, car les supports de leurs valeurs n'existent pas encore;

-Fait correspondre les lignes (N + NBEX +1) à (N + NBEX + p) de statut chacune (2), aux variables références (si on représente les valeurs des références sur deux éléments minimums).

La file tableau occupe ainsi,(NBEX + P + 1) lignes de la configuration,représentées sur un support de (NBLI*T + 2*P +6) éléments minimums de la file support, où "T" ést la taille du statut d'une ligne du tableau ("T" = e1+e2+..+en).

Le code généré correspondant à la phrase de déclaration est:

(DEC.TAB,N,NBLI,NBEX,NBLE,e1,e2,...,en)

et pour chaque référence Ri

(DEC.REF,N+NBEX+i,2)

(*) Les six premiers champs de la ligne "N" sérvent pour la géstion du débordement et l'accès aux éléments du tableau; ils contiennent respectivement:

-Le type de la file : "tableau";

-Le nombre de lignes de la variable initiale : NBLI;

-Le nombre total des lignes du tableau incrémenté à chaque extension de "NBLE"; : Initialement NBLI

-Le nombre d'extensions possibles : NBEX;

-Le nombre de lignes d'une variable d'extension : NBLE;

-Le nombre d'extensions déjà faites : Initialement zéro;

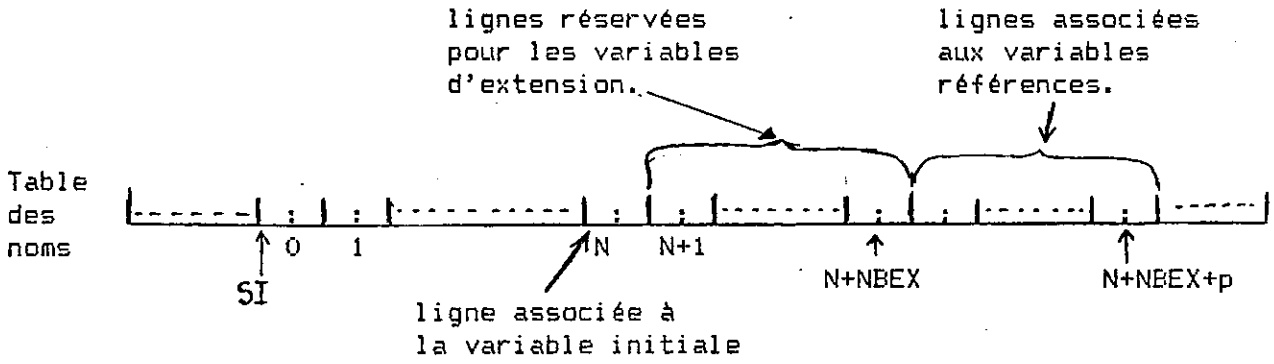


FIGURE 3 : REPRESENTATION DU TABLEAU SUR LA CONFIGURATION

TABLES DE COMPILATION

Pour pouvoir compiler du texte source et produire son image, le générateur a besoin de certaines informations qu'il doit garder dans des tables, au fur et à mesure de la lecture du texte.
On va expliciter quelle est l'information à conserver pour le tableau.

La table des références

pour chaque référence :
- Son nom;
- Son numéro de ligne associé dans la configuration;
- Une entrée dans la table des files pour reconnaitre à quelle file elle est attachée.

La table des files

pour chaque file :

- Son type : "tableau"
- Son numéro de ligne associé à la variable initiale : N
- La taille d'une ligne de la file : (e1+e2+..+en)
- Le nombre d'éléments d'une ligne : n
- Pour chaque élément : . Son nom : eli
  . Sa taille : ei
- Le nombre de lignes de la variable initiale : NBLI
- Le nombre d'extensions possibles : NBEX

Dans la suite, on va expliciter pour chacune des phrases à compiler, le code généré correspondant et l'information récupérée par le générateur de ces tables.

UTILISATION DU TABLEAU

a) Ri := initial

Le générateur récupère de la table des références, le numéro de ligne NRi associé Ri et génére: (:=TAB,NRi,"initial")

b) Ri :=ultime

Le générateur récupère des deux tables:

NRi : Le numéro de ligne associé à Ri

N : Le numéro de ligne associé à la variable initiale

et génére : (:=TAB,NRi,N,"ultime")

c) Ri := exp ---> Rj ; Ri := exp <--- Rj

Le générateur vérifie que les références Ri et Rj sont attachées à la même file,

récupère des deux tables:

NRi : Le numéro de ligne associé à Ri

NRj : Le numéro de ligne associé à Rj

N : Le numéro de ligne associé à la variable initiale

T : La taille d'une ligne du tableau (e1+e2..+en)

et génére selon le cas :

(:=-->TAB,NRi,NRj,exp,N,T,e1,e2,..,en)

ou

(:=<--TAB,NRi,NRj,exp)

d) $\Gamma_{Ri,elj}$

Le générateur vérifie que la référence Ri est bien attachée à une file qui contient l'élément elj,

récupère des deux tables:

N : Le numéro de ligne associé à la variable initiale

NRi : Le numéro de ligne associé à Ri

NBC : Le nombre d'éléments d'une ligne tableau (n)

et génére : (elt.TAB,N,NRi,j,NBC)

e) Γ_{Ri}

Le générateur récupère des deux tables:

NRi : Le numéro de ligne associé à Ri

N : Le numéro de ligne associé à la variable initiale

NBC : Le nombre d'éléments d'une ligne tableau

et génére : (lig.TAB,N,NRi,NBC)

LE COMPILATEUR DE DEROULEMENT

Dans le contexte du dérouleur on déclare une file support qui contiendra les configurations des procédures déroulées:

support TAILLE ,NBLIGNES,H(1) :S1,S2,S3,S4,S5,SI

Les références S1,S2,S3,S4 ET S5 servent pour le parcours de la file. La référence SI repère l'origine de la configuration,elle est positionnée sur la première ligne de la configuration de la procédure qu'on est entrain de traiter.Cette ligne est réservée pour le débordement et se compose de trois éléments (on verra plus loin son utilité).

DEBORDEMENT DU TABLEAU

Le nombre de lignes d'un tableau étant détérminé à la déclaration, les fonctions initial et ultime délimitent l'ensemble des valeurs que peut prendre une référence attachée à ce tableau.

Si on affecte à une référence une valeur plus petite que celle donnée par initial(Cf.EXP1) ou une valeur plus grande que celle donnée par ultime (Cf.EXP2),il y a débordement du tableau

EXP1	EXP2
Y := initial	Y := ultime
Y := 1 <--- Y	Y := 1 ---> Y
DEBORDEMENT A GAUCHE	DEBORDEMENT A DROITE

Si on a un "débordement à droite" du tableau et qu'à la déclaration on avait prévu des extensions (Cf.FIGURE 4);dans la première ligne de la configuration, le dérouleur met:

- dans dans son premier élément, le numéro de la ligne "N" associée à la variable initiale pour signaler l'extension au système;ce numéro permet d'identifier le tableau à étendre;
- dans son deuxième élément, le nombre d'éléments minimums par lesquels le tableau va être étendu;

ensuite donne la" main" au système.

Le système alloue (par l'instruction ligne) une ligne de la file support dont la taille est fournie par le dérouleur (deuxième élément de la première ligne de la configuration) et met dans le troisième élément de la première ligne de la configuration,la distance (en nombre de lignes) qui sépare cette dérnière de la ligne qui vient d'être allouée.

Au prochain quantum alloué pour l'execution du programme image,le dérouleur remet à jour les données concérnées par l'extension et continue le traitement normal du code.

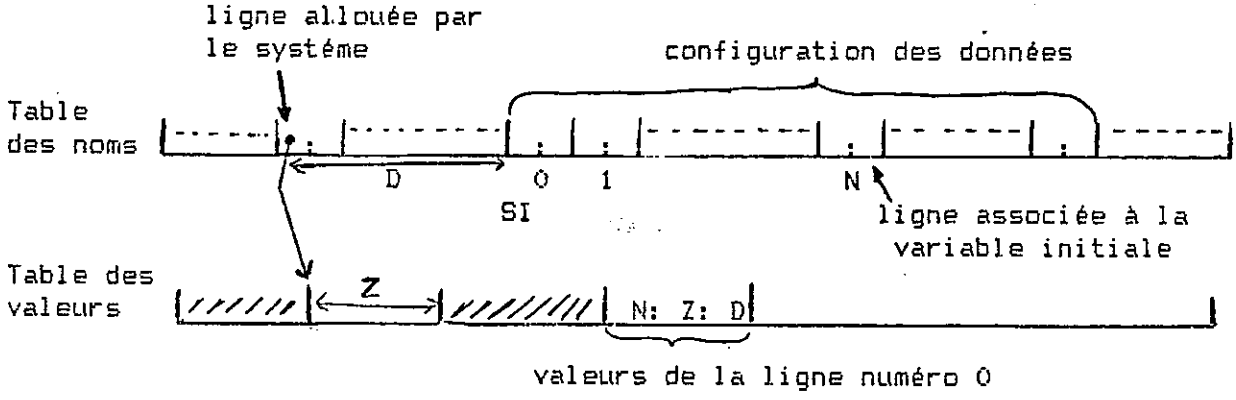


FIGURE 4 :SCHEMA DE LA FILE SUPPORT EN CAS D'EXTENSION DU TABLEAU.

DEROULEMENT DES INSTRUCTIONS IMAGES

Je suppose que le code généré se trouve dans une file message (chaque ligne contient une instruction image), la lecture du code se fait par une référence "PC" positionnée sur la ligne de l'instruction à traiter.Pour la clarté de l'exposé, je travaillerai directement sur les valeurs du code,sans oublier que dans une vérsion définitive ces valeurs sont à remplacer par les "instructions de lecture" des éléments correspondants dans la file message.

(DEC.TAB,N,NBLI,NBEX,NBLE,e1,e2,....,en)

Création de la ligne "N" associée à la variable initiale

S1 := N ---> SI	S1 est positionnée sur la ligne associée à la variable initiale.
H := 6*2 + NBLI*(e1+e2+...+en)	Calcul de la taille de la ligne "N".
LIGNE S1,H	
[S1 := STATUT (6(2),NBLI(e1,e2,...,en))	
	Caractéristiques de la file:
[S1,1 := "Tableau"	Type de la file;
[S1,2 := NBLI	Nombre de lignes de la variable initiale;
[S1,3 := NBLI	Nombre total des lignes du tableau;
[S1,4 := NBEX	Nombre d'extensions possibles;
[S1,5 := NBLE	Nombre de lignes par lesquelles on étend le tableau;
[S1,6 := 0	Nombre d'extensions éfféctuées;
PC := 1 --->PC	Passer à l'instruction suivante.

(DEC,REF,NRi,2)

Création de la ligne NRi associée à la référence Ri

S1 := NRi ---> SI

H := 2

LIGNE S1,H

[S1 := STATUT (2)

PC := 1 --->PC

(:=TAB,NRi,"initial")

"Ri := initial"

S1 := NRi ---> SI

[S1 := 1

PC := 1 ---> PC

(:=TAB,NRi,N,"ultime")

"Ri := ultime"

S1 := NRi ---> SI

S2 := N ---> SI

[S1 := [S2,3

PC := 1 ---> PC

(:=---->TAB,NRi,NRj,exp,N,I,e1,e2,.....en) "Ri :=exp ---> Rj"

S1 := NRi ---> SI S1 est positionnée sur la ligne associée à la référence Ri

S2 := NRj ---> SI S2 est positionée sur la ligne associée à la référence Rj

S3 := N ---> SI S3 est positionnée sur la ligne associée à la variable initiale

AIGUILLAGE maj (⌈SI,1 = N,E1) (⌈SI,1 <> N,E2)

E1 : Une demande d'extension a été faite pour ce tableau,mise à jour des données concérnées par cette extension

INSERER M.A.J.EXT
SORTIE maj

E2 : SORTIE maj
NOEUD maj

AIGUILLAGE bornesup (⌈S2 <= ⌈S3,3 - exp ,E3) (⌈S2 >⌈S3,3 - exp ,E4)

E3 : La valeur affectée à Ri est inferieure à la borne suppérieure du tableau,traitement normal.

⌈S1 := ⌈S2 + exp
PC := 1 ---> PC
SORTIE bornesup

E4 : La valeur affectée à Ri est plus grande que la borne superieure, traitement du débordement à droite.

AIGUILLAGE extens (⌈S3,6 < ⌈S3,4 ,E5) (⌈S3,6 = ⌈S3,4 ,E6)

E5 : Le nombre d'extensions éffectuées sur le tableau est inférieur à "NBEX" (nombre maximum d'extensions),traitement de l'extension.

⌈SI,1 := N Signaler l'extension au système;

⌈SI,2 := ⌈S3,5 * T Fournir la taille de la zone à allouer et on reste sur la même instruction;

SORTIE extens

E6 : Le nombre d'extensions éffectuées est égal à "NBEX",traitement de l'erreur du débordement;

INSERER ERREUR DEBORDEMENT
SORTIE extens

NOEUD extens
SORTIE bornesup

NOEUD bornesup

PROCEDURE M.A.J.EXT

Cette procédure remet à jour les caractéristiques de la file et fait coïncider "le support des valeurs" d'une ligne réservée pour l'extension avec celui de la ligne allouée par le système.

DEBUT

⌊SI,1 := 0 — Annuler le signal d'extension;

⌊S3,3 := ⌊S3,3 + ⌊S3,5 — Mise à jour du nombre total des lignes du tableau;

⌊S3,6 := ⌊S3,6 + 1 — mise à jour du nombre d'extensions éffectuées

S4 := ⌊S3,6 ---> S3 — S4 est positionnée sur la ligne réservée pour l'extension;

AIGUILLAGE droi.gauch (⌊SI,3 < 0 ,E7) (⌊SI,3 > 0 ,E8)

E7 : La ligne allouée par le système est à droite de la référence SI

S5 := ⌊SI,3 ---> SI

SORTIE droi.gauch

E8 : La ligne allouée par le système est à gauche de SI

S5 := ⌊SI,3 <--- SI

SORTIE droi.gauch

NOEUD droi.gauch — S5 est positionnée sur la ligne allouée par le système

LIGNE S4,S5 — On fait coïncider "les supports des valeurs" des deux lignes

⌊S4 := STATUT (⌊S3,5 (e1,e2,...,en))

FIN

(:=<---TAB,NRi,NRj,exp) "Ri := exp <--- Rj"

S1 := NRi ---> SI

S2 := NRj ---> SI

AIGUILLAGE borneinf (⌈S2 > exp ,E9) (⌈S2 <= exp ,E10)

E9 : La valeur affectée à Ri est superieure à la borne inferieure du tableau

⌈S1 := ⌈S2 - exp

PC := 1 ---> PC

SORTIE borneinf

E10 : La valeur affectée à Ri est plus petite que la borne inferieure

INSERER ERREUR DEBODEMENT

SORTIE borneinf

NOEUD borneinf

La procédure ERREUR DEBORDEMENT arrête l'exécution du programme et sort un message d'erreur.

(elt,TAB,N,NRi,NBC,j) ; (lig,TAB,N,NRi,NBC)

" ⌈Ri,elj" " ⌈Ri"

Je rappelle la phrase de déclaration:

tableau NBLI (el1(e1),...,eln(en)) NBEX EXT NBLE :R1,...,Rp

Le tableau est représenté par la variable initiale et les variables d'extension par lesquelles on l'a étendu.La variable initiale est une ligne de la file support de statut (6(2),NBLI(e1,..en),elle se compose donc de (6+NBLI*n) champs,les six premiers champs contiennent les caractéristiques du tableau,les autres

champs pris par paquets de "n" représentent les lignes du tableau.

Une variable d'extension est une ligne de la file support de statut (NBEX (e1,...,en)) dont chaque paquet de "n" champs représente une ligne du tableau.

Ces paquets sont ordonnés,c'est à dire que les paquets de la variable initiale représentent les lignes de un à NBLI du tableau,les paquets de la première variable d'extension représentent les lignes de (NBLI + 1) à (NBLI + NBLE) du tableau,ect...

Section commune pour les deux codes :

S1 := NRi --->SI

S2 := N --->SI

AIGUILLAGE varINIT.varEXT (⌊S1<= ⌊S2,2 ,ET1)(⌊S1 > ⌊S2,2 ,ET2)

Auiguillage suivant que la ligne du tableau se trouve dans la variable initiale ou dans une variable d'extension.

ET1 : La ligne se trouve dans la variable initiale

S3 := S2	S3 est positionnée sur la ligne de la file support associée à la variable initiale
h :=(⌊S1 -1)*NBC +6 +1	h est la correction à faire dans la ligne S3 pour accèder au premier champ du paquet.
SORTIE varINIT.varEXT	

ET2 : La ligne se trouve dans une variable d'extension

l := ⌊S1 - ⌊S2,2	l := le nombre de lignes du tableau repésentées par les variables d'extension (⌊S1 - NBLI)
q := l DIV ⌊S2,5	q := le quotient de la division entière de l par NBLE
r := l MOD ⌊S2,5	r := le réste de la division
S3 := (q+1)---> S2	S3 est positionnée sur la ligne associée à la variable d'extension qui contient la ligne du tableau

h := (r - 1)*NBC +1 h := la correction à faire dans la ligne S3 pour accèder au premier champ du paquet

SORTIE varINIT.varEXT

NOEUD varINIT.varEXT

On a donc dans S3 la ligne de la file support qui contient la ligne du tableau et dans "h" la correction à faire dans cette ligne pour accèder au premier champ du paquet.

Si on a le code correspondant à " ⌊Ri,elj" on accède à l'élément du tableau par : " ⌊S3,(h +j - 1)".
Si on a le code correspondant à " ⌊Ri" on accède à la ligne du tableau par une boucle de "n" itérations de ⌊S3,h jusqu'à ⌊S3,(h + n - 1) (pour lire tout le paquet).

DEROULEUR DU LAC

écrit en

Procédure Formelle Symbolique

p. sanchez

C.R. Subject classification informatics D34 B14

Résumé.

Dans le cadre de l'étude des Systèmes, le présent article décrit la réalisation d'un compilateur de déroulement du LAC écrit en P.F.S.

L'écriture de ce compilateur dans un langage machine évolué comme la P.F.S. aide à la mise au point de ce langage et permet de vérifier que son utilisation résoud les problèmes posés par l'écriture du système et notamment les problèmes posés par l'autojectivité des compilateurs.

COMPILATEUR DE DEROULEMENT DU LAC ECRIT EN PROCEDURE FORMELLE SYMBOLIQUE

Ce travail s'inscrit dans le cadre d'une recherche sur les compilateurs autojectifs, pouvant s'insérer dans un système multi-utilisateur.
Il convient donc de donner une définition de l'autojectivité pour les compilateurs.

"Un compilateur autojectif est tel qu'il possède un seul point d'entrée et un seul point de sortie. Tout déroulement décrit un parcours unique fini borné qui l'emmène du point d'entrée au point de sortie. Il faudra donc controler, en particulier, les boucles qui sont les seuls éléments susceptibles de ne pas appartenir au domaine du fini borné".

La commutation avec le système et les différents compilateurs ne pourra se faire qu'en des points prédéterminés.

Nous allons avant de décrire le compilateur de déroulement, faire un rappel sur le langage LAC, sur son compilateur, sur la forme du code généré et sur le système minimal dans lequel s'insèreront les différents compilateurs et autres procédures du système nécessaire au déroulement.

LANGAGE LAC

Le LAC comprend une partie déclaration ou figure le nom de la procédure et les différentes déclarations de variables, de paramètres et d'index, caractérisée par les lettres:

pro, par, var, ind.

La lettre deb annonce le debut des instructions et la fin des déclarations.
Les instructions se terminent par la lettre ";".

Chaque opération utilise au maximum trois opérandes avec:

- 4 opérateurs arithmétiques :=+,:=*;:=-,:=/
- 5 opérateurs logiques avec saut conditionnel
  si=,si>,si<,si>=,si<= (vers etiquette)
- l'affectation :=
- le saut inconditionnel vers suivi d'une étiquette.

Le ":" permet de ponctuer les étiquettes.
On peut également insérer des procédures grâce à l'instruction ins suivi du nom de la procédure à insérer et de la liste des paramètres.

EXEMPLE DE PROGRAMME ECRIT EN LAC

```
pro CAL;                        pro FCT;
var F,P,T(10);                  par A,N;
ind I;                          var C;
deb;                            deb;
I:=4;                           si N=0 vers El;
F:=I;                           A:=A*N;
P:=F-1;                         C:=N-1;
ins FCT(F,P);                   ins FCT(A,C);
T(I):=F;                    El: fin;
fin;
```

L'étude du compilateur LAC a déjà été faite dans le bulletin n° 4 (-BULLETIN D'INFORMATIQUE APPROFONDIE ET APPLICATIONS- Publication du Département de Mathématique et Informatique, Décembre 1982, "AUTOJECTION ET COMPILATION". E. BIANCO) et l'écriture en PFS reste à faire. Néanmoins comme toute phase de déroulement débute par une phase de compilation, il faut connaître les caractéristiques du compilateur générant pour comprendre le déroulement.
Le compilateur LAC écrit en PFS n'étant pas encore réalisé, pourquoi ne pas lui donner toutes les commodités qui faciliteront l'écriture du compilateur de déroulement.

GENERALITES SUR LE SYSTEME

Le système est composé de manière générale par:

- le superviseur
- le compilateur de langage hors texte
- un triplet composé du compilateur du langage évolué, du compilateur du langage de traitement des données, du compilateur du langage de commandes.

Le superviseur distribue des quanta de travail aux différents compilateurs et controle leurs déroulements. C'est la seule partie non autojective du système. Dans le cas le plus simple c'est une boucle.

Le langage hors texte permet de choisir le compilateur avec lequel on désire travailler, par exemple le LAC.

Les compilateurs des langages de programmation transforment une phrase symbolique en une autre phrase susceptible d'être utilisée par les compilateurs de déroulement, eux même appliquant cette phrase sur des données fabriquées par des langages de traitement de données et donnant des résultats à l'usage de l'utilisateur.

- les compilateurs d'échange prennent en charge les demandes d'échanges entre les différents compilateurs et établissent la correspondance entre ceux-ci et les périphériques.

FILES UTILISEES

Le système utilise une file support sur laquelle se placeront les paquets d'informations que représentent les différents compilateurs et les configurations de données.

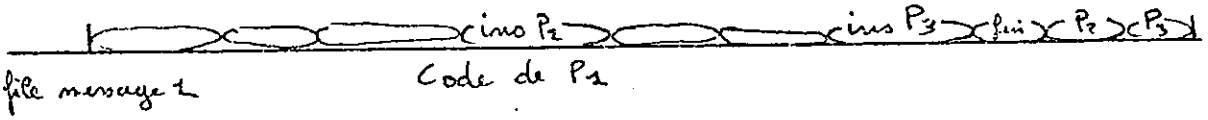
Bien entendu nous supposons que la machine universelle de la PFS existe.

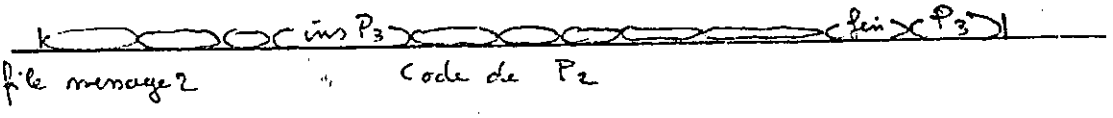
Les codes des différentes procédures sont sur des files messages. Pendant la génération du code, le compilateur LAC remplit une file message. A la fin, si cette file est incomplète, une opération de réduction permet au système de ne conserver que des files messages pleines. En effet à la fin de la compilation le compilateur connaît la longueur de

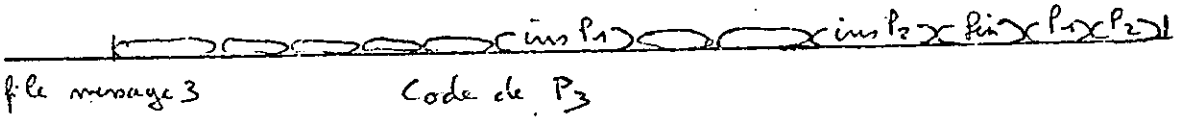
code qu'il vient de générer. Il connaît aussi la longueur de la configuration de données nécessaire au déroulement. Cette information devra figurer dans le code pour permettre au compilateur déroulant de créer cette zone.
Lors des commandes de déroulement, l'utilisateur créera sa zone de données. Cette zone est pour le système une file message dont chaque élément représentera une configuration. Au moment de l'insertion d'une nouvelle procédure le compilateur de déroulement placera à la suite sur la file message un nouvel élément dont le statut, lu dans le nouveau code, sera différent.

LA CONSTRUCTION DU PROGRAMME

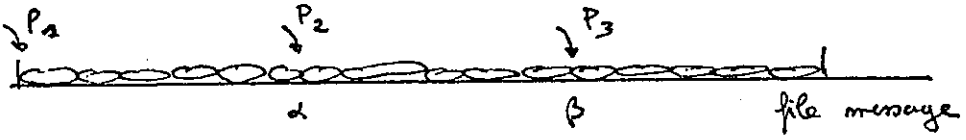
Le système fixe une file message pour la construction du code du programme, dont la longueur peut être donnée par l'utilisateur.
Ensuite chaque code de procédure se juxtapose au précédent sur la file et une opération du langage transforme les deux files messages en une file unique. Le processus se reproduisant à chaque procédure constituant le programme. Le système possède dans un tableau l'indication lui permettant de retrouver les origines de tous les codes des différentes procédures constituant le programme.
Après la compilation des procédure P1, P2, P3 nous obtenons les trois files messages suivantes:







L'opération de construction du programme transforme les trois files messages en une seule file message et construit un tableau d'accès au debut du code de chaque procédure. L'origine du code des différentes procédures est repérée par rapport à l'origine de la file message.



| P1 , ∅ | P2 , α | P3 , β |

Bien entendu la procédure de construction du programme est beaucoup plus complexe car elle doit vérifier que les paramètres sont compatibles pour l'insertion. Les informations pour réaliser ce travail se trouvent dans la section globale des files messages de chaque procédure et dans le code, à la fin, avec le nom des procédures à insérer.

GENERALITE SUR LE COMPILATEUR LAC GENERANT

Pour chaque procédure compilée, le compilateur LAC fournit au système une file message dont la longueur sera connue à la fin de la compilation
A la fin de la compilation le système connaît également la file message qui contient le code et la taille de la configuration de données nécessaire pour le déroulement.
Le système gère sur sa file support l'ensemble des files messages résultant de la compilation des procédures.

Nous allons nous interresser au code généré par la compilation d'une phrase LAC, car l'écriture du compilateur de déroulement et sa simplicité dépendent essentiellement de la clarté de ce code.
Pour que le déroulement puisse avoir lieu il faut aussi définir une configuration de données.

CONFIGURATION DE DONNEES

Le système alloue à chaque utilisateur une place sur sa file support dont les caractéristiques seront fournies au compilateur de déroulement par l'intermédiaire du langage de communication.
Pour le déroulement le compilateur à besoin d'une file message pour recevoir les différentes configurations et nous supposerons que chaque élément de la file sera rattaché à une procédure et aura la structure suivante:

adres. retour nom par. nom var. nom const. valeurs

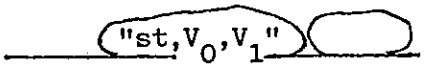
les noms seront formés, pour une variable ou un paramètre, de deux cases dont nous verrons la fonction un peu plus loin.

On se donne les files messages suivantes:

```
message ??(r1,r2,r3:1) cod;
message ??(h,h1,h2,h3:1) conf,conf1,conf2,conf3;
```

qui contiendront respectivement le code et les configurations.
Les points d'interrogations seront remplis par le système d'une part grâce au compilateur générant et d'autre part grâce au langage de communication.
A l'intérieur de la configuration un élément sera repéré par la référence qui pointe sur l'origine du paquet et par le nombre de cases qui le sépare de cette origine.
Pour une variable locale on réservera deux cases dans la partie nom des variables locales, la première case contiendra le déplacement que doit effectuer la référence (0 dans ce cas précis), la deuxième case contiendra la valeur de l'index h qui permet à partir de la position de la référence d'accéder à la h^{ieme} case en respectant le statut de la configuration.
Dans le cas d'un paramètre la première case contient le déplacement vers la gauche de la référence qui pointera

ainsi sur l'élément de la file contenant la valeur du paramètre.
Nous verrons tout cela plus en détail par la suite.
La première instruction que l'on rencontrera dans le code est une instruction faisant référence au statut global de la configuration de données, statut qui diffèrera d'une procédure à l'autre selon les déclarations et le corps du programme.

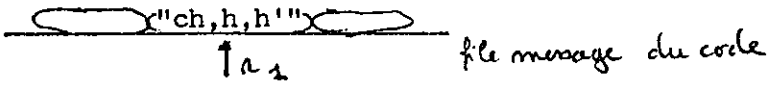


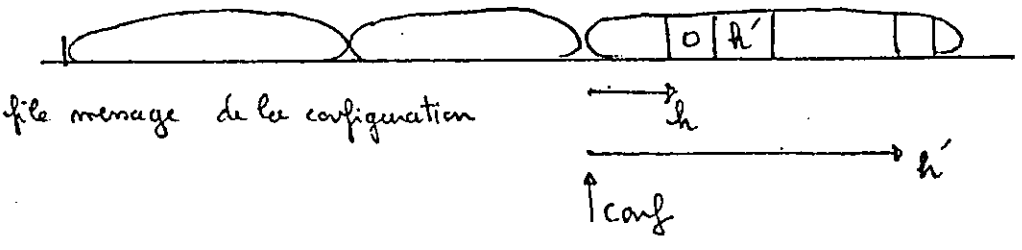
V_0 : volume des noms = 2*(par+var+ind+cte)
chaque case de la zone des noms a pour longueur 2 octets
V_1 = volume des valeurs, sur 4 octets.

Précisons maintenant les propriétés du compilateur générant, notamment au niveau du code.

VARIABLE

Dans le code nous trouverons deux parties, la partie déclaration et la partie traitement.
Dans les déclarations nous saurons qu'à la rencontre du code "ch" il faudra réserver deux cases à la distance h de l'origine de la configuration et que dans la première case nous placerons le déplacement vers la gauche de la référence, dans la deuxième la distance h' ou nous trouverons la valeur.



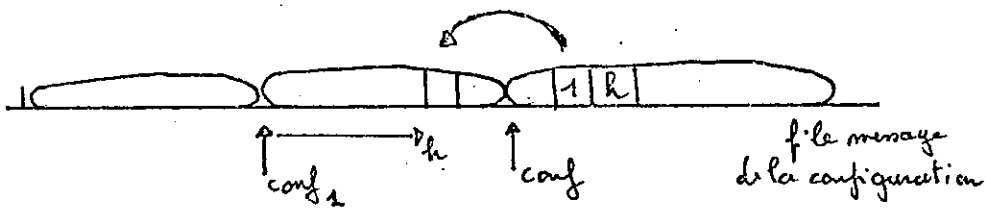


Partie du compilateur de déroulement qui traitera ce problème:

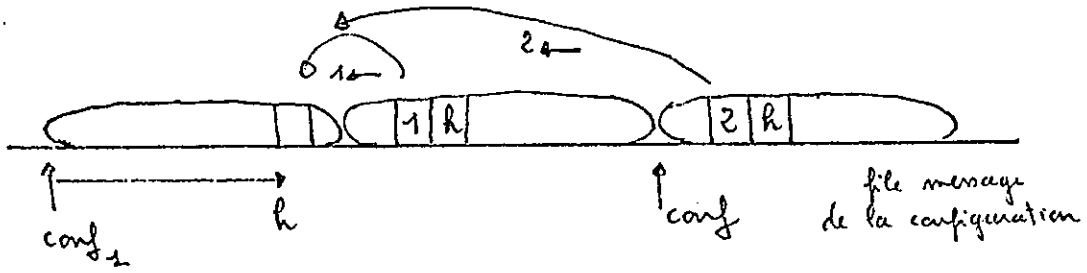
```
h1 := C cod,r1 ;          !h1=h,1ere case
C conf1,h1 := 0 ;         !var. de la conf.
h1 := h1 + 1 ;            !2eme case
r1 := r1 + 1 ;            !on prend h
C conf1,h1 := C cod,r1 ;  !h' dans 2eme case
```

PARAMETRE

Le compilateur générant réserve les deux cases qui seront remplies au moment de l'insertion de la manière suivante:
dans le cas de l'insertion d'une variable locale on place dans la première case correspondante de la configuration insérée la valeur 1 qui représente le déplacement vers la gauche que doit éffectuer la référence lors du changement de configuration pour retrouver dans l'élément précédent la valeur de la variable, et dans la deuxième case nous trouverons la distance par rapport à l'origine de la configuration insérante.

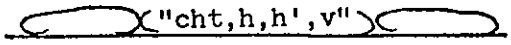


Dans le cas de l'insertion d'un paramètre on duplique dans la configuration insérée les deux cases attachées au paramètre en incrémentant la première case pour que la référence se déplace d'une configuration supplémentaire sur la gauche.



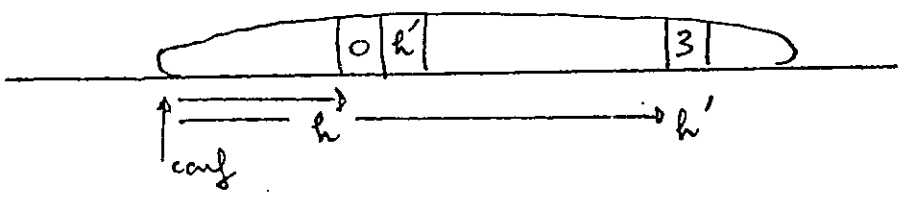
CONSTANTE

Après la rencontre de la lettre "deb" par le compilateur LAC lorsque celui-ci trouvera une constante il génèrera le code suivant:



ou "cht" indique que l'on procède au chargement d'une variable de type constante,
"h" donne le rang de la première case de la zone des noms de la configuration, case que l'on initialisera à zéro (déplacement de la référence),
"h'" donne le rang de la case ou figurera la valeur de la constante par rapport au debut de la configuration en respectant les statuts,
"v" represente la valeur de cette constante que nous placerons au moment du déroulement.
Ainsi lors du déroulement la constante est traitée comme une variable locale.

$h1 := C_{cod,r1}$;	!h1=h
$C_{conf,h1} := 0$;	!valeur dans la conf
$h1 := h1 + 1$;	!case suivante
$r1 := r1 + 1$;	!positionne h'
$C_{conf,h1} := C_{cod,r1}$;	!place h' dans case
$h2 := C_{conf,h1}$;	!positionne valeur
$r1 := r1 + 1$;	!prend la valeur
$C_{conf,h2} := C_{cod,r1}$;	!range la valeur



CAS D'UNE VARIABLE NON INDICEE

On défini une section telle que l'on n'ait qu'à passer des paramètres, le calcul du positionnement des références se faisant à l'intérieur de la section.

codx : valeur de la référence qui pointe sur le premier élément du code.

rx : indice courant dans l'élément du code correspondant.

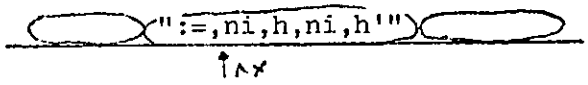
confi : valeur de la référence qui pointe sur la configuration en cours.

hx : indice courant dans l'élément de la configuration correspondant.

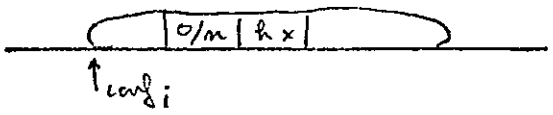
Nous supposerons que dans le code, le type de la variable est immédiatement suivi de l'indication permettant de la retrouver dans la configuration.

Pour une variable non indicée le type est noté "ni".

Exemple :




```
section NIND ( codx,rx,confi,confx,hx ) ;
rx := rx + 1 ;
hi := C codx,rx ;
confx := C confi,hi ← confi ;
hi := hi + 1 ;
hx := C confi,hi ;
rx := rx + 1 ;
```



CAS D'UNE VARIABLE INDICEE

Dans le cas d'une variable indicée nous supposerons que le code du type de la variable est immédiatement suivi des indications permettant de retrouver la variable et son indice dans la configuration.

Exemple : ":=,i,h,h',ni,h'"

```
section IND ( codx,rx,confi,confx,hx ) ;
rx := rx + 2 ;
hi := C codx,rx ;
confx := C confi,hi <- confi ;
hi := hi + 1 ;
hx := C confi,hi ;
hi' := C confx,hx ;
rx := rx - 1 ;
hi := C codx,rx ;
confx := C confi,hi <- confi ;
hi := hi + 1 ;
hx := C confi,hi + hi' ;
rx := rx + 2 ;
```

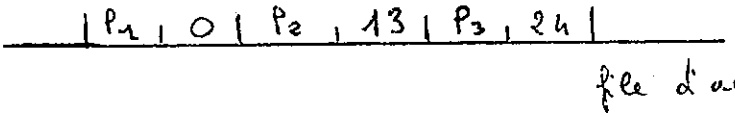
CAS DE L'INSERTION DES PROCEDURES

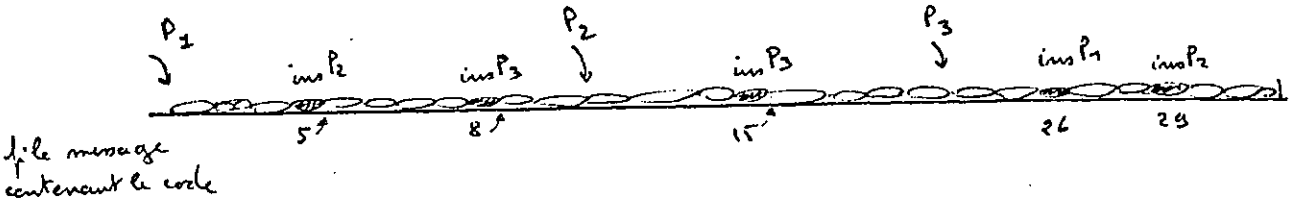
Nous allons voir que l'insertion de procédure est essentiellement une opération du système.
Nous supposerons qu'au moment de l'assemblage définitif d'un programme, c'est à dire pendant le "linkage" le compilateur d'échange fournira au système une file qui contiendra la liste des noms des différentes procédures constituant le programme, suivi d'une indication permettant d'atteindre le code. Cette indication pouvant être le déplacement de la référence de la file support.
Ainsi au moment de l'insertion d'une procédure figurant dans la liste des procédures présentes dans la file "linkage", le compilateur de déroulement prépare la configuration suivante en déplacant vers la droite la référence qui pointait dans la configuration appelante et sauvegarde quelque part dans la configuration appelée la référence de retour dans la file du code de la procédure appelante.
Le passage des paramètres s'étant fait comme vu précédemment le système reprend le controle et à la nouvelle insertion du compilateur de déroulement le travail reprendra avec les

nouvelles valeurs des références.

Le déroulement sera terminé lorsque ayant rencontré un code "fin", le compilateur lira, dans la case contenant le retour dans le code de la procédure appelante, un retour systématique au système.

Lorsque le compilateur du langage de commande donnera au système le nom de la première procédure qui appelera toutes les autres, la construction du programme débutera par la mise en place du code de cette procédure au début de la file message allouée au code ,et la première valeur dans le tableau d'accès sera un zéro. A partir de la, le déplacement pour le calcul des origines des autres procédures se fait grâce à la connaissance du nombre d'élément du code de chaque procédure.
Le système placera dans la première case de la file message réservée au déroulement un code d'arrêt.
A la rencontre de l'instruction d'insertion, après le passage correct des paramètres, il faudra placer au début de chaque configuration insérée le retour a l'élément suivant le code d'insertion. Puis avant de déplacer l'origine de la configuration d'un élément vers la droite, il faudra chercher grâce au nom de la procédure à insérer la nouvelle origine dans le code et passer ces nouveaux paramètres au système qui réalisera l'insertion.
La rencontre du mot "fin" dans le code déclanchera automatiquement la scrutation de la première case de la configuration. Si la valeur est différente du code d'arrêt le système aura immédiatement le saut de retour dans le code, et l'on déplacera cette fois-ci l'origine de la configuration d'un élément vers la gauche.
Un petit exemple pour éclairer les explications.





aspect au cours du déroulement

Le programme qui suit a pour but de familiariser avec le langage de la P.F.S., et sa réalisation a permi de mieux cerner les problèmes du déroulement du code des instructions LAC, tout en respectant les contraintes de la structure autojective du compilateur.

```
procedure COMPILATEUR (adpro, cod, conf) ;

message ??(r1,r2,r3:1) cod:

message ??(h,h',h1,h2,h3,hi:1)conf,conf1,conf2,conf3,confi:
                              1 ( 1, m(2), n(4) ),
                              2 ( 1, p(2) );

refpar cod,conf,adpro,nompro ;
refvar conf1,conf2,conf3,confi ;
var h,h',h1,h2,h3,hi,r1,r2,r3,t ;

          debut ;
          cod := init ;
          cod := C adpro -> cod ;
          r1 := 1 ;

          aiguillage CODE (si C cod,r1 =("st":e1,"ch":e2,"cht":e3,
          "aff":e4,":=+":e5,":=-":e5,":=*":e5,":=/":e5,"si=":e6,
          "si>":e6,"si<":e6,"si>=":e6,"si<=":e6,"ins":e7,"fin":e8 ))

          ----------statut de la configuration----------

e1:       r1 := 2 ;
          r2 := 3 ;
          m := C cod,r1              !nb. de cases des noms
          n := C cod,r2              !nb. de cases des valeurs
          conf := conf, statut 1     !nouveau statut de la conf.
          C adpro := C adpro + 1 ;   !élément suivant
          sortie CODE ;

          ----------chargement d'une variable----------

e2:       r1 := 2 ;
          h := C cod,r1 -> conf ;    !on pointe le nom
          C conf,h := 0 ;            !la var. est locale
          r1:= 3 ;                   !pointe l'adresse dans le code
          h := h + 1 ;               !deuxième case du nom
```

```
        C(conf,h) := C(cod,r1) ;      !met l'adresse dans la case
        C(adpro) := C(adpro) + 1      !élément suivant
        sortie CODE ;

        ----------chargement d'une constante----------

e3:     r1 := 2 ;
        h := C(cod,r1) -> conf ;      !pointe la première case du nom
        C(conf,h) := 0 ;              !la constante est locale
        r1 := 3 ;                     !dans le code, l'adresse de la valeur
        h := h + 1 ;                  !deuxième case du nom
        C(conf,h) := C(cod,r1) ;      !on place l'adresse
        h' := C(conf,h) ;             !pointe l'adresse de la valeur
        r1 := 4 ;                     !dans le code, la valeur
        C(conf,h') := C(cod,r1) ;     !place la valeur dans la conf.
        C(adpro) := C(adpro) + 1 ;    !élément suivant
        sortie CODE ;

        ----------affectation----------

e4:     r1 := 2 ;
        aiguillage VAR1 (si C(cod,r1) =("i":e41,"ni":e42)) ;

e41:    section IND (cod,r1,conf,conf1,h1) ;
        sortie VAR1 ;

e42:    section NIND (cod,r1,conf,conf1,h1) ;
        sortie VAR1 ;
        noeud VAR1 ;
```

On a positionné conf1 et h1 du premier opérande

```
        aiguillage VAR2 (si C(cod,r1) =("i":e43,"ni":e44)) ;

e43:    section IND (cod,r1,conf,conf2,h2) ;
        sortie VAR2 ;

e44:    section NIND (cod,r1,conf,conf2,h2) ;
```

```
        sortie VAR2 ;
        noeud VAR2 ;
```

On a positinné conf2 et h2 du deuxième opérande

```
        C_conf1,h1 := C_conf2,h2 ;
        C_adpro := C_adpro + 1 ;   !élément suivant
        sortie CODE ;

        ----------operateur arithmétique----------

e5:     r1 := 2 ;                  !on teste le type
        aiguillage OPA1 (si C_cod,r1 =("ni":e51,"i":e52)) ;

e51:    section NIND.(cod,r1,conf,conf1,h1) ;
        sortie OPA1 ;

e52:    section IND (cod,r1,conf,conf1,h1) ;
        sortie OPA1 ;
        noeud OPA1 ;

        aiguillage OPA2 (si C_cod,r1 =("ni":e53,"i":e54)) ;

e53:    section NIND (cod,r1,conf,conf2,h2) ;
        sortie OPA2 ;

e54:    section IND (cod,r1,conf,conf2,h2) ;
        sortie OPA2 ;
        noeud OPA2 ;

        aiguillage OPA3 (si C_cod,r1 =("ni":e55,"i":e56)) ;

e55:    section NIND (cod,r1,conf,conf3,h3) ;
        sortie OPA3 ;

e56:    section IND (cod,r1,conf,conf3,h3) ;
        sortie OPA3 ;
        noeud OPA3 ;
```

```
        r1 := 1 ;                    !on teste le code opératoire
        aiguillage OPA (si C cod,r1 =(":=+":op1,":=-":op2, ":=/":op3,
                        ":=*":op4)) ;

op1:    C conf1,h1 := C conf2,h2 + C conf3,h3 ;
        sortie OPA ;

op2:    C conf1,h1 := C conf2,h2 - C conf3,h3 ;
        sortie OPA ;

op3:    C conf1,h1 := C conf2,h2 / C conf3,h3 ;
        sortie OPA ;

op4:    C conf1,h1 := C conf2,h2 * C conf3,h3 ;
        sortie OPA ;
        noeud OPA ;

        C adpro := C adpro + 1 ;   !élément suivant
        sortie CODE ;

        ----------opérateur logique--------------------

e6:     r1 := 2;
        aiguillage OPL1 (si C cod,r1 =("ni":e61,"i":e62) ;

e61:    section NIND (cod,r1,conf,conf1,h1) ;
        sortie OPL1 ;

e62:    section IND (cod,r1,conf,conf1,h1) ;
        sortie OPL1 ;
        noeud OPL1 ;

        aiguillage OPL2 (si C cod,r1 =("ni":e63,"i":e64) ;

e63:    section NIND (cod,r1,conf,conf2,h2) ;
        sortie OPL2 ;
```

```
e64:        section IND (cod,r1,conf,conf2,h2) ;
            sortie OPL2 ;
            noeud OPL2 ;

            r2 := r1 ;
            r1 := 1 ;
            aiguillage OPL (si C cod,r1 =("si=":opl1,"si ":opl2, "si ":opl3,
                            "si =":opl4,"si =":opl5 ) ;

op11:       aiguillage TOP1 (si C conf1,h1 = C conf2,h2 :etop1 sinon etop11) ;

op12:       aiguillage TOP2 (si C conf1,h1 > C conf2,h2 :etop1 sinon etop11) ;

op13:       aiguillage TOP3 (si C conf1,h1 < C conf2,h2 :etop1 sinon etop11) ;

op14:       aiguillage TOP4 (si C conf1,h1 >= C conf2,h2 :etop1 sinon etop11) ;

op15:       aiguillage TOP5 (si C conf1,h1 <= C conf2,h2 :etop1 sinon etop11) ;

etop1:      r2 := r2 + 1 ;
```

Dans le cas ou la déclaration d'étiquette se fait avant ou après le vers la référence se déplace vers la droite ou vers la gauche.

```
            aiguillage DG (si C cod,r2 =( "g":eg, "d":ed)) ;
ed:         r2 := r2 + 1 ;
            C adpro := C adpro + C cod,r2 ;
            sortie DG ;

eg:         r2 := r2 + 1 ;
            C adpro := C adpro - C cod,r2 ;
            sortie DG ;
            noeud DG ;

            sortie ( TOP5, TOP4, TOP3, TOP2, TOP1 ) ;
            sortie OPL ;

etop11:     C adpro := C adpro + 1 ;
```

```
        sortie (TOP5, TOP4, TOP3, TOP2, TOP1 ) ;
        noeud ( TOP5, TOP4, TOP3, TOP2, TOP1 ) ;

        sortie OPL ;
        noeud OPL ;
        sortie CODE ;
```

----------insertion----------

```
e7:     r2 := 2 ;
        r1 := 3 ;
        p := C(cod,r2) ;
        confi := 1 -> conf, statut 2 ;        !on ne connait que le
        hf := C(cod,r2) ;                     !statut des noms
        itere BOUC (hi de 1 a hf pas 2 ) ouverture 5;
        h := C(cod,r1) -> conf ;
        C(confi,hi) := C(conf,h) + 1 ;        !passage des paramètres
        h := h + 1 ;
        hj := hi + 1 ;
        C(confi,hj) := C(conf,h) ;
        r1 := r1 + 1;
        repetition BOUC ;
        conf := confi ;                       !deplace origine de la conf.
        r3 := 1 ;
        C(conf,r3) := C(adpro) + 1 ;
        C(nompro) := C(cod,r1) ;
        section RECHPRO ( nompro, adpro ) ;
```

Procédure qui en fonction du nom de la procédure à insérer qui se trouve dans le code donne la nouvelle valeur de adpro.

```
        sortie CODE ;
```

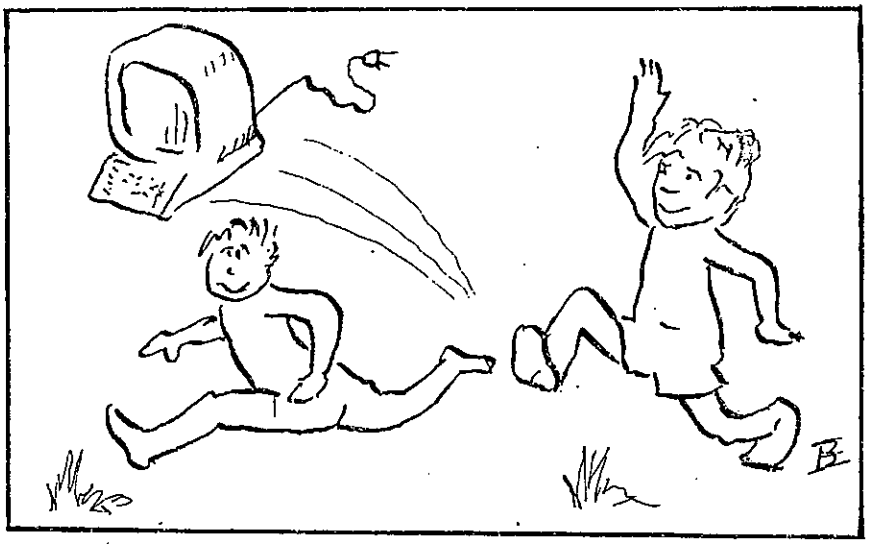
----------fin----------

```
e8:     h := 1 ;
```

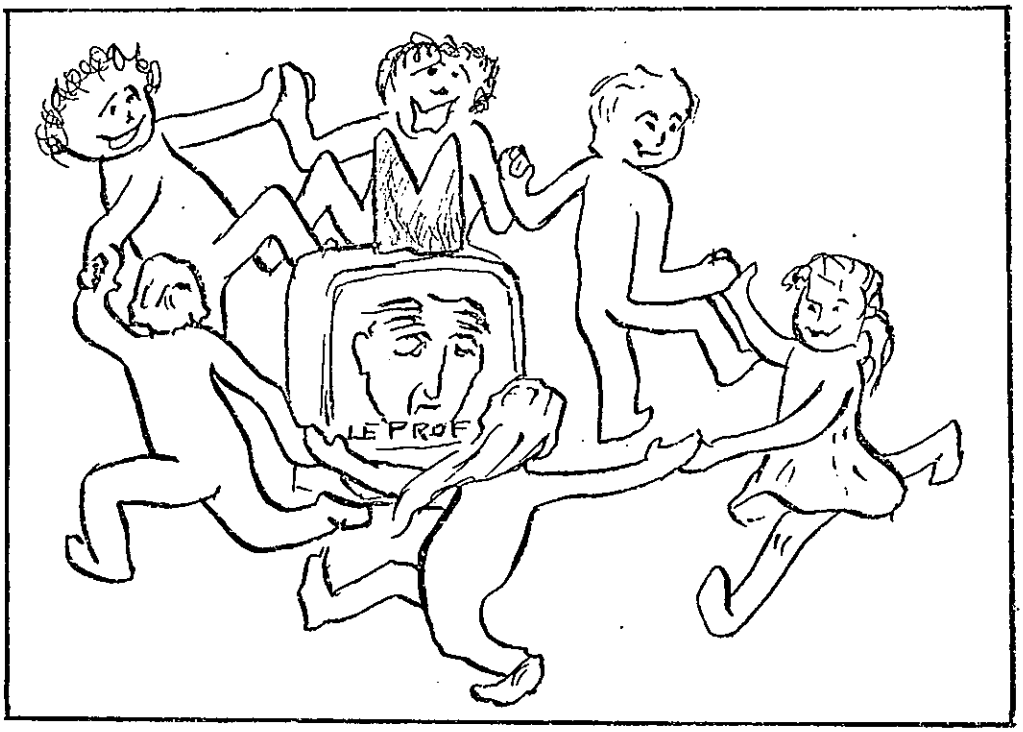
```
        aiguillage FINI (si C conf,h =("f":ef1 sinon ef2);

ef1:    fin ;
        sortie FINI ;
ef2:    C adpro := C conf,h ;
        conf := 1 <- conf ;        !deplace vers la gauche l'origine
                                   !de la configuration

        sortie FINI ;
        noeud FINI ;
        sortie CODE ;
        noeud CODE ;
```

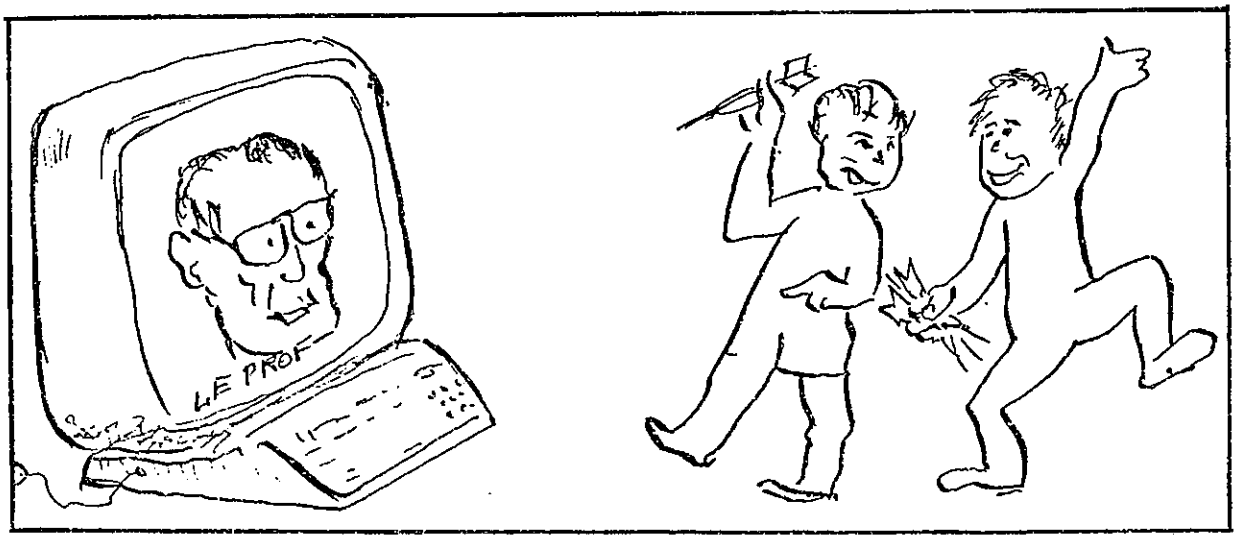
VOUZZAVEDIBISAR.

L'Ordinateur à l'Ecole.



Ça vaut pas mon ballon, mais c'est bien plus marrant !





Il a dit : 'droit au but !'