Chez Prof McJones,

Avec mes compliments,

V. Dahl

groupe d'intelligence artificielle

uer scientifique de luminy
70 route léon-lachamp
13009 marseille

UN SYSTEME DE BANQUE DE DONNEES EN LOGIQUE DU PREMIER ORDRE, EN VUE DE SA CONSULTATION EN LANGUE NATURELLE

V. Dahl

R. Sambuc

université d'aix-marseille II

RAPPORT PRESENTE EN VUE

DE L'OBTENTION DU

D.E.A. D'INTELLIGENCE ARTIFICIELLE

---

le 30 septembre, 1976

Nous tenons à remercier vivement Alain COLMERAUER, Maître de Conférence en Informatique, qui a dirigé ce travail, pour le temps qu'il a consacré à de fructueuses discussions et pour ses précieux conseils qui nous ont permis de mener notre recherche à son terme, ainsi que tous les membres du Groupe d'Intelligence Artificielle, pour le climat dans lequel celle-ci s'est déroulée.

1 - INTRODUCTION.

Ce travail constitue la deuxième partie d'une recherche entreprise dans le Groupe d'Intelligence Artificielle de l'U.E.R. Scientifique de LUMINY, à MARSEILLE, sur la réalisation d'une banque de données en logique du premier ordre, consultable en français.

La première partie, qui fera l'objet d'un autre rapport, réalise l'interface entre les questions posées par l'utilisateur et les formules logiques constituant l'entrée du programme décrit ici.

L'application particulière choisie a été la représentation d'un sous-ensemble du catalogue du matériel et du logiciel du système SOLAR 16, réalisée d'une telle façon qu'à partir de chaque demande de l'utilisateur sera engendré, par assemblage des divers éléments compatibles entre eux, le système qui convient à la demande.

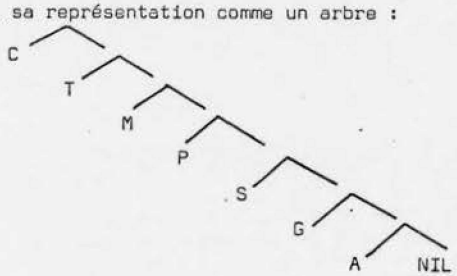
Indépendamment de l'exemple pris, les résultats obtenus devraient être utilisables pour le problème des banques de données en général.

2 - DESCRIPTION DE LA BANQUE DE DONNEES.

Les objets les plus généraux que l'on traite dans la banque de données sont des systèmes. Un système est défini par l'ensemble des éléments suivants :

- une configuration de base (C)
- une taille de mémoire (T)
- une liste des modules de mémoire (M)
- une liste de périphériques (P)
- une liste de logiciel de base (S)
- une liste de langages (G)
- une liste de modules optionnels (A)

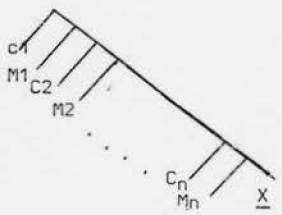
Voici sa représentation comme un arbre :



Cet arbre n'est pas construit séquentiellement, mais dans l'ordre qui se dégage de la question posée par l'utilisateur. Par exemple, si celui-ci exige un système avec le langage FORTRAN, ce langage est immédiatement ajouté dans la liste de langages, et l'arbre sera complété par la suite.

Pour obtenir cela, deux conventions ont été adoptées :

i - Chaque liste de modules (c'est-à-dire M, P, G, S et A) a la structure suivante :



où C_i $(1 \leq i \leq n)$ représente un cardinal, et

M_i $(1 \leq i \leq n)$ représente un module, et $M_i \neq M_j$

Si $i \neq j$ (chaque module n'apparaît qu'une fois dans la liste). La variable X qui apparaît à la fin permet de rallonger la liste et sera unifiée à NIL une fois que celle-ci aura été complétée (fermeture de la liste).

ii - Pour que les cardinaux puissent être changés au fur et à mesure que les différentes fonctions ajoutent des modules dans la liste, on a choisi la représentation suivante :

0 sera écrit, au départ : X

n sera écrit, au départ : X'''...'' où le nombre de "'" est égal à n.

Donc, si à un moment déterminé de l'exécution il faut, par exemple trois modules SMX12 et qu'il n'y en ait que deux, la fonction chargée de cette vérification n'aura qu'à transformer le 2 en 3, par simple unification des variables.

(X'' sera unifié contre Y''', donc X sera remplacé par Y' et le nouveau cardinal figurant dans la liste sera Y''').

La taille de mémoire respecte les mêmes conventions, donc au départ elle prendra la valeur minimum compatible avec l'unité centrale considérée, et s'agrandira en fonction des exigences des autres modules du système.

Pour limiter un certain cardinal, il suffit donc d'unifier sa variable à zéro pour qu'il ne puisse plus croître. Au moment de la fermeture d'une liste, tous ses cardinaux subissent ce traitement.

Voici un exemple d'un système complet :

```
CARACTERISTIQUES DU SYSTEME DEMANDE:
====================================

- CONFIGURATION DE BASE: BC.T.40, OPTION: 0
- TAILLE DE LA MEMOIRE CENTRALE: 32 K MOTS
- STRUCTURE DES MODULES DE MEMOIRE:

2  FCM16

- LISTE DE PERIPHERIQUES :

1  ASR33

- SOFTWARE :

1  BOSD

- LANGAGES :

1  APLS1

- MODULES OPTIONNELS :

1  ASMD
1  EDILE
1  EDITEX
1  MACP
```

3 - DESCRIPTION DU LANGAGE D'ENTREE.

Les formules logiques acceptées sont construites à partir des prédicats définis dans la banque de données (c'est-à-dire, des fonctions PROLOG) qu'on représentera désormais par les symboles $\underline{p}$ ou $\underline{p}_i$ $(\forall\ i \in N)$.

Si $\underline{p}$ ou $\underline{p}_i$ sont des formules acceptables et $\underline{x}$ une variable PROLOG, les formules suivantes sont également acceptables :

i - TOUT ($\underline{x}$, $\underline{p}$) (Pour tout $\underline{x}$, $\underline{p}$)

ii - EXISTE ($\underline{x}$, $\underline{p}$) (Il existe un $\underline{x}$ tel que $\underline{p}$)

iii - NON ($\underline{p}$)

iv - ET ($\underline{p}_1$, $\underline{p}_2$) ($\underline{p}_1$ et $\underline{p}_2$)

v - OU($\underline{p}_1$, $\underline{p}_2$) ($\underline{p}_1$ ou $\underline{p}_2$)

vi - IMPL ($\underline{p}_1$, $\underline{p}_2$) ($\underline{p}_1$ implique $\underline{p}_2$)

En fait, la première occurrence de toute variable quantifiée est précédée de l'étiquette LAMBDA. Ainsi, les règles i) et ii) sont en réalité :

i - TOUT (LAMBDA($\underline{x}$,$\underline{p}$))

ii - EXISTE (LAMBDA($\underline{x}$,$\underline{p}$))

Ces formules logiques subissent d'abord un pré-traitement pour leur simplification. Par exemple, "non(et($\underline{p}_1$, $\underline{p}_2$))" sera transformé en "IMPL($\underline{p}_1$,NON($\underline{p}_2$))".

4 - CARACTERISTIQUES DE LA BASE DE DONNEES.

4.1 - Non déterminisme.

La base de données n'est pas définie seulement par des clauses unaires (comme, par exemple, "LANGAGE(FORTRAN)", mais aussi par des procédures définissant toutes les versions potentielles du monde qu'elle représente.

Chaque demande de l'utilisateur engendrera dynamiquement l'une de ces versions, en fonction des contraintes exprimées dans la question.

4.2 - Exploitation prévue.

L'exploitation de cette banque de données se limite à la consultation. Les modifications sur les données qui s'y trouvent ne peuvent donc pas être faites sans comprendre les mécanismes du programme.

4.3 - Indépendance vis-à-vis de l'organisation physique.

Puisque la définition de la banque a été faite en termes de la logique du premier ordre, où les chemins d'accès et l'organisation physique n'interviennent pas (étant gérés par l'interpréteur PROLOG), on peut la considérer, pour l'exploitation prévue, comme indépendante vis-à-vis de l'organisation physique des données.

4.4 - Langage de manipulation des données.

Il n'est pas défini par des primitives accessibles seulement au programmeur, mais constitue un sous-ensemble assez vaste du français.

La base de données est ainsi non seulement indépendante de l'organisation physique, mais aussi des clauses particulières qui composent le programme.

4.5 - Fonction de sélection.

Indépendamment des différents ordonnements et paraphrases possibles pour la même question, les clauses du programme s'exécutent dans l'ordre correct, grâce à un autre trait fondamental du système : l'inclusion d'un démonstrateur permettant de définir sa propre fonction de sélection en vue d'altérer l'ordre séquentiel d'exécution des prédicats.

Par exemple, le prix d'un système ne peut pas être calculé avant que le système ait été complètement engendré. Or, l'utilisateur peut très bien formuler la question :

(1) "Donnez-moi le prix d'un système qui possède BASIC".

Si l'on se limitait à ordonner les appels aux différents prédicats à partir de la question posée, le prédicat PRIX(x, y) (qui définit le prix de x comme égal à y) serait exécuté avant le prédicat SYSTEME(x), ce qui conduirait à un échec.

La solution adoptée consiste à définir pour chaque prédicat un "retard" compris entre 0 et une valeur maximum. On peut ainsi définir pour un prédicat plusieurs retards, correspondant à des conditions différentes que le démonstrateur testera dynamiquement lors de l'exécution.

Par exemple, le prédicat PRIX (x, y) n'aura aucun retard si la variable x a été complètement instanciée (*), et aura le retard maximum dans le cas contraire.

Ainsi, suivant la valeur de ce retard, le prédicat est alors soit exécuté, soit mis dans une liste d'attente.

Ce trait est exploité aussi en vue de l'efficacité : en reprenant l'exemple (1), si on le suivait textuellement, il faudrait engendrer un système arbitrairement, puis, une fois construit, vérifier s'il possède BASIC et retourner en arrière s'il se trouve qu'il ne le possède pas.

Le nombre de systèmes possibles étant fini mais très grand, on a intérêt à inclure d'abord BASIC dans la liste des langages du système, et à le compléter après, d'une manière compatible avec l'existence de BASIC. Cela est simple à obtenir en définissant des retards appropriés pour les prédicats SYSTEME et POSSEDE. Dans l'Annexe 2 se trouve un exemple du fonctionnement du démonstrateur.

Bien sûr, le fait d'exécuter le démonstrateur pour chaque demande ralentit la démonstration. Cela est inévitable pour les prédicats qui ont besoin d'être retardés, mais a été évité pour les autres prédicats, qui seront immédiatement exécutés au lieu d'être ajoutés à la liste à examiner par le démonstrateur.

Finalement, il convient de souligner que le démonstrateur est indépendant de la banque de données en particulier, et que ses applications possibles ne se limitent pas aux systèmes de banques de données : il s'agit, en effet, d'un sous-programme permettant de définir soi-même la fonction de sélection choisie, d'une façon dynamique.

(*) Il convient de faire ici la distinction entre une variable telle que la logique l'entend, et une variable PROLOG : cette dernière peut être liée à un terme contenant d'autres variables libres. Dans ce cas, on dit qu'elle n'a pas été complètement instanciée.

4.6 - Traitement de la négation.

Dans le cadre du langage PROLOG, les propriétés négatives sont définies par défaut. Par exemple, si dans un univers de quatre individus on définit deux d'entre eux comme étant des étudiants, les deux autres seront considérés implicitement comme non étudiants. D'une façon générale, si p est une formule logique quelconque, NON(p) sera évaluée à VRAI si l'évaluation de p fait échec, et à FAUX si l'évaluation de p est possible. Cela implique que pour exécuter le NON il faut d'abord exécuter son argument, ce qui pose un problème qu'on tentera d'expliciter au moyen de l'exemple suivant :

Considérons un univers d'individus défini par les clauses :

INDIVIDU (Pierre)

INDIVIDU (Marie)

INDIVIDU (Jeanne)

ELEVE (Jeanne)

L'évaluation du prédicat ELEVE(x) aura comme résultat l'unification de x à "Jeanne".

Si l'on veut connaître la liste des individus qui ne sont pas élèves, on peut faire les appels correspondants de deux façons possibles :

i - INDIVIDU(x) . NON(ELEVE(x)).

ii - NON(ELEVE(x)) . INDIVIDU(x) .

Dans le premier cas, x est unifié d'abord à Pierre, ensuite la clause NON(ELEVE(Pierre)) est évaluée à VRAI, puisque ELEVE(Pierre) fait échec, donc Pierre est la première réponse trouvée. Marie sera la deuxième réponse, et pour Jeanne le NON fera échec.

Par contre, dans le deuxième cas, l'évaluation de la première clause ne conduira à aucun succès, donc la deuxième ne sera jamais exécutée : aucune réponse n'est possible.
En effet, puisque l'évaluation du NON suppose celle de son argument, et que dans le cas considéré cette évaluation a comme unique résultat l'unification de x à Jeanne et l'évaluation à VRAI de la clause ELEVE(x), la clause NON(ELEVE(x)) fera échec.

On voit donc clairement que l'ordre d'exécution des prédicats parmi lesquels il y a des négations est fondamental.

Une solution possible consisterait à ramener systématiquement à la fin d'une suite d'appels tous les prédicats négatifs. Mais il serait intéressant de pouvoir se servir des clauses négatives pour limiter le champ de recherche de la solution. Par exemple, la demande d'un système ne possédant pas FORTRAN devrait interdire l'apparition de ce langage avant d'engendrer le système, au lieu de vérifier son absence une fois le système construit.

Pour résoudre ce problème, on se sert de la fonction de sélection décrite précédemment, en retardant l'exécution des prédicats négatifs jusqu'à ce que les variables libres dans la portée du NON aient été instanciées (plus précisément, substituées par des termes ne contenant pas de variables libres).

- ANNEXE 1 -

EXPLICATION DU DEMONSTRATEUR

- Superdémontrer (L)

  lance la démonstration de la formule logique L, en commençant par les prédicats de priorité 0.

- Démontrer (N, L)

  démontre successivement les prédicats de priorité N, N+1, ... etc, qu'il y a dans L.

- Exec (N, L1, L2, M)

  démontre les prédicats de priorité N qu'il y a dans L1, concatène dans L2 les prédicats de priorité plus haute que N et rend dans M la prochaine priorité à considérer.

  Pour les prédicats négatifs (reconnus par l'étiquette NIET), calcule la liste deleursvariables libres et les traite par le prédicat EXECO.

  Chaque fois qu'un prédicat a été cancelé, M sera mis à 0 (le résultat de la cancelation ayant pu satisfaire pour quelque prédicat les conditions sous lesquelles son retard est nul).

  Quand les listes L1 et L2 ont été épuisées pour une priorité quelconque, on passe à la priorité suivante, pourvu que la priorité actuelle ne soit pas la maximum admise, ce qui indiquerait une erreur.

- Execo (L1, L2, M)

  si la liste de variables liées à l'extérieur du prédicat négatif à démontrer (le premier dans L1) est vide, lance sa démonstration récursivement (en employant le démonstrateur lui-même). Autrement, le concatène dans la liste d'attente L2.

  M joue le même rôle que dans EXEC.

- Variables (X, Y)

  rend dans Y la liste de variables libres qui apparaissent dans l'expression X, au moyen du prédicat VARS.

- Vars (L, Y0, Y1, Y2)

  garde dans Y0 la liste des variables liées dans l'expression L, dans Y1 la liste ancienne des variables liées à l'extérieur (c'est-à-dire, libres dans L), et dans Y2 la nouvelle liste de ces variables.

  Appelle VARBIS pour chaque élément dans L.

- Varbis (A, Y0, Y1, Y)

  examine A : s'il s'agit d'une variable qui est déjà dans Y1 (ancienne liste des variables liées à l'extérieur), la nouvelle liste Y sera égale à l'ancienne; si c'est une variable qui n'y est pas, elle sera concaténée dans Y. S'il s'agit d'une variable interne (détectée par l'étiquette LAMBDA qui la précède), elle sera concaténée dans la liste Y0. Finalement, s'il s'agit d'une fonction (1), elle sera découpée par UNIV et ses arguments seront examinés par VARS.

- Dans (A, Y)

  le résultat sera VRAI si A se trouve dans Y et FAUX autrement.

- Retardmax (N)

  N définit le maximum de retard à considérer.

- Not (L)

  sera évalué à VRAI si la formule L est fausse, et à FAUX sinon.

- Conc (X, Y, Z)

  rend dans Z la concaténation de la liste X avec la liste Y.

- Retard (P, R)

  définit le prédicat P comme ayant un retard égal à R.

- Eg (X, Y)

  unifie X contre Y.

(1) Les constantes sont considérées comme des fonctions de rang 0.

- ANNEXE 2 -

EXEMPLE DU FONCTIONNEMENT DU DEMONSTRATEUR

Dans l'exemple suivant on a demandé un système avec une taille de memoire de 24 K mots, et on a fait imprimer la trace des prédicats exécutés successivement.

Les prédicats EXISTE, ET et SYS s'exécutent dans l'ordre dans lequel ils apparaissent, puisqu'ils ont tous les trois un retard égal à zéro. Le seul effet de l'exécution de SYS est de garder dans la liste d'attente du démonstrateur l'appel au prédicat SYS1, qui construit le système.

Comme le retard de SYS1 a été défini comme égal a 2, tandis que le prédicat TAILLE a 0 comme retard, c'est ce dernier qui s'exécute en premier, fixant ainsi la taille a 24 K mots, et ce n'est qu'après que le système est completé par SYS1.

```
SUPERLEMONTRER(EXISTE(LAMBDA(*X,ET(SYS(*X),TAILLE(*X,24))))).
    ~      'SORTIR(*X)..

EXISTE(LAMBDA(*X1,ET(SYS(*X1),TAILLE(*X1,24))))
ET(SYS(*X1),TAILLE(*X1,24))
SYS(*X1)
TAILLE(SYS(*X1),24)
DIVEX(24,4,*X1)
TRANSF(6,*X1)
SYS1(*X1.*X2.*X3.*X4.TAI.*X5'''''.*X6.*X7.*X8.*X9.*X10.NIL)
LISTLAN(*X1,*X2,*X3'''''',*X4)
LISTLAN1(*X1,*X2,*X3'''''',*X4)
LSOPT(*X1,*X2'''''',*X3,*X4,*X5,*X6)
LSOPT1(*X1,*X2'''''',*X3,*X4,*X5,*X6)
COL(*X1.*X2.*X3,*X4'''''',*X5,*X6,*X7,*X8)
LISTEDEB(*X1,*X2,*X3,*X4)
INFEG(*X1,0)
INFEG(*X1,0)
INFEG(*X1,16)
NMAXIM(*X1'''''',9)
MEMO(*X1,5,0'''''')
DEP(5,*X1,0'''''')
POSSEDE1(*X1,0'.NIL)
INFEG(0'''''',8)
FERMNBRE(0'''''')
FERMLST(*X1)
FERMLST(*X1)
FERMLST(*X1''.SNB04.*X2'.SMY16.*X3'.SMY04.*X4)
PLACEAL(*X1.*X2.5.*X3,0''.SNB04.0'.SMY16.0'.SMY04.NIL,NIL,*X4)
```

CARACTERISTIQUES DU SYSTEME DEMANDE:
==========

- CONFIGURATION DE BASE: BC.T.5, OPTION: C
- TAILLE DE LA MEMOIRE CENTRALE: 24 K MOTS
- STRUCTURE DES MODULES DE MEMOIRE:

2 SME04
1 SMY12
1 SMM04

- LISTE DE PERIPHERIQUES :

- SOFTWARE :

- LANGAGES :

- MODULES OPTIONNELS :

- ANNEXE 3 -

EXEMPLES D'UTILISATION DE LA BASE DE DONNEES

- Un système avec une taille de mémoire de 16 K mots, qui ou bien ne possède pas le langage Basic, ou bien possède le langage Fortran:

```
SUPERDEMONTRER(EXISTE(LAMBDA(*X,ET(SYS(*X),ET(TAILLE(*X,16),
      OU(NON(POSSEDE(*X,0'.BASC)),POSSEDE(*X,0'.FOR16)))))).
     SORTIR(*X)..
```

```
CARACTERISTIQUES DU SYSTEME DEMANDE:
====================================

- CONFIGURATION DE BASE: BC.T.S, OPTION: 0
- TAILLE DE LA MEMOIRE CENTRALE: 16 K MOTS
- STRUCTURE DES MODULES DE MEMOIRE:

1   SMB04
1   SMM12

- LISTE DE PERIPHERIQUES :

- SOFTWARE :

- LANGAGES :

- MODULES OPTIONNELS :
```

- Un système possédant le langage FOR16:

```
SUPERDEMONTRER(EXISTE(LAMBDA(*X,ET(SYS(*X),POSSEDE(*X,'.FOR16))))).
        SORTIR(*X)..


CARACTERISTIQUES DU SYSTEME DEMANDE:
====================================

- CONFIGURATION DE BASE: BC.T.40, OPTION: 0
- TAILLE DE LA MEMOIRE CENTRALE: 16 K MOTS
- STRUCTURE DES MODULES DE MEMOIRE:

1   FCM16

- LISTE DE PERIPHERIQUES :

1   ASR33

- SOFTWARE :

1   BASL

- LANGAGES :

1   FOR16

- MODULES OPTIONNELS :

1   ASMD
1   EDILE
1   EDITEX
1   MACR
```

- Un système avec une unité centrale égale à 65 et son prix:

```
SUPERDEMONTRER(EXISTE(LAMBDA(*X,ET(SYS(*X),ET(UNITCENT(*X,65),
      EXISTE(LAMBDA(*P,PRIX(*X,*P))))))))). SORTIR(*X). SORP(*P)..


CARACTERISTIQUES DU SYSTEME DEMANDE:
====================================

- CONFIGURATION DE BASE: EC.B.65, OPTION: 0
- TAILLE DE LA MEMOIRE CENTRALE: 32 K MOTS
- STRUCTURE DES MODULES DE MEMOIRE:

2  FCM16

- LISTE DE PERIPHERIQUES :

- SOFTWARE :

- LANGAGES :

- MODULES OPTIONNELS :


- PRIX DU SYSTEME :184000 FRANCS


SOPTAX(*P): ..
```

- Un système qui possède le processeur optionnel MTS05 et le module de mémoire SMX08, et dont la taille de mémoire soit différente de 12 K mots:

```
SUPERDEMONTREP(EXISTE(LANGLA(*Y),ET(SYS(*Y),ET(POSSEDE(*Y,0'.MTS05),
        ET(POSSEDE(*Y,0'.SMX08), NON(TAILLE(*Y,12)))))))).
        SORTIR(*Y)..
```

```
CARACTERISTIQUES DU SYSTEME DEMANDE:
====================================

- CONFIGURATION DE BASE: DC.T.5, OPTION: 0
- TAILLE DE LA MEMOIRE CENTRALE: 16 K MOTS
- STRUCTURE DES MODULES DE MEMOIRE:

1  SMX08
2  SMD04

- LISTE DE PERIPHERIQUES :

- SOFTWARE :

- LANGAGES :

- MODULES OPTIONNELS :

1  MTS05
```

- ANNEXE 4 -

LISTING DU PROGRAMME

```
** (1) DEMONSTRATEUR..

SUPERDEMONTRER(*L): DEMONTRER(0,*L.NIL)..

DEMONTRER(0,NIL): //..
DEMONTRER(*N,*L): EXEC(*N,*L,*L2,*M). DEMONTRER(*M,*L2)..

EXEC(0,NIET(*Y,*L).*L1,*L2,*M): //. VARIABLES(*L,*V).
        EXEC0(NIET(*Y,*V).*L1,*L2,*M)..
EXEC(*N,*X.*L1,*L3,0): RETARD(*Y,*N1). EG(*N1,*N). //.
        AX(*X,*L2). CONC(*L2,*L1,*L3)..
EXEC(*N,*X.*L1,*X.*L2,*M): EXEC(*N,*L1,*L2,*M)..
EXEC(*N,NIL,NIL,*M): NOT(RETARDMAX(*N)). //. PLUS(1,*N,*M)..
EXEC(*N,NIL,NIL,*M): SORT("ERREUR").IMPASSE..

EXEC0(NIET(*Y,NIL).*L1,*L1,0): //.NOT(DEMONTRER(0,*Y))..
EXEC0(*X.*L1,*X.*L2,*N): EXEC(0,*L1,*L2,*M)..

VARIABLES(*X,*Y): VARS(*X,NIL,NIL,*Y)..

VARS(NIL,*Y0,*Y1,*Y1): ..
VARS(*A.*X,*Y0,*Y1,*Y2): VARBIS(*A,*Y0,*Y1,*Y).VARS(*X,*Y0,*Y,*Y2)..
VARBIS(*A,*Y0,*Y1,*Y1): VAR(*A). DANS(*A,*Y0). //..
VARBIS(*A,*Y0,*Y1,*Y1): VAR(*A). DANS(*A,*Y1). //..
VARBIS(*A,*Y0,*Y1,*A.*Y1): VAR(*A). //..
VARBIS(LAMBDA(*X,*P),*Y0,*Y1,*Y2): //.VARS(*P.NIL,*X.*Y0,*Y1,*Y2)..
VARBIS(*A,*Y0,*Y1,*Y2): UNIV(*A,*B.*X). VARS(*X,*Y0,*Y1,*Y2)..

DANS(*A,*B.*X): EGAL(*A,*B). //..
DANS(*A,*B.*X): DANS(*A,*X)..

RETARDMAX(5): ..

NOT(*L): *L. //.IMPASSE..
NOT(*L): ..

CONC(NIL,*Y,*Y): ..
CONC(*A.*X,*Y,*A.*V): CONC(*X,*Y,*V)..

RETARD(PRIX(*Y,*Y),0): VARIABLES(*X.NIL,NIL). //..
RETARD(PRIX(*X,*Y),4): //..
RETARD(SYS1(*X),2): //..
RETARD(POSSEDE(*X,*Y),1): //..
RETARD(TAILLE(*X,*T),1): //..
RETARD(CONFBASE(*X,*C),1): //..
RETARD(UNITCINT(*X,*U),1): //..
RETARD(OPTUC(*X,*O),1): //..
RETARD(SORTIR(*X),5): //..
RETARD(*X,0): ..

EG(*X,*X): ..
```

```
**  AX(*P,*Y): UNIFIE *Y AVEC LA SUITE D'APPELS CORRESPONDANT A
              LA FORMULE LOGIQUE *P..


AX(*P,*Y): SOFTAX(*P). IMPASSE..
AX(EXISTE(LAMBDA(*X,*P)),*P.NIL): //..
AX(TOUT(LAMBDA(*X,*P)),NON(EXISTE(LAMBDA(*X,NON(*P)))).NIL): //..
AX(ET(*P1,*P2),*P1.*P2.NIL): //..
AX(OU(*P1,*P2),*P1.NIL): ..
AX(OU(*P1,*P2),*P2.NIL): //..
AX(NON(NON(*DIEU)),*DIEU.NIL): //..
AX(NON(ET(*P1,*P2)),IMPL(*P1,NON(*P2)).NIL): //..
AX(NON(IMPL(*P1,*P2)),ET(*P1,NON(*P2)).NIL): //..
AX(NON(TOUT(LAMBDA(*X,*P))),EXISTE(LAMBDA(*X,NON(*P))).NIL): //..
AX(NON(POSSEDE1(*X,*N.*Y)),POSSEDE1(*X,0.*Y).NIL): //..
AX(NON(*P),NIET(*P.NIL,*P.NIL).NIL): //..
AX(IMPL(*P1,*P2),OU(NON(*P1),*P2).NIL): ..


** (2) BANQUE DE DONNEES..


AX(PRIX(NIL,0),NIL): // ..
AX(PRIX(SYS(*X.*Y.*U.*O.*S),*F),PRIX1(*X.*Y.*U.*O,*W). PRIX(*S,*Z)
. PLUS(*W,*Z,*F). NIL): //..
AX(PRIX(TAI.*T.*L,*F),PRIX(*L,*F).NIL): //..
AX(PRIX(*C1.*M1.*L,*P), TRANSF(*V,*C1).PRIX1(*M1,*F1)
          .MULT(*V,*F1,*F2).PRIX(*L,*F).PLUS(*F2,*F,*P).NIL):
           NUMER(*C1).//..
AX(PRIX(*L1.*L,*F),PRIX(*L1,*F1).PRIX(*L,*F2).PLUS(*F1,*F2,*F)
          .NIL): //..
AX(PRIX(*X,*F),NIL) : PRIX1(*X,*F). //..
AX(SYS(SYS(*X)),SYS1(*Y).NIL): ..

AX(SYS1(*X.*Y.*U.*O.TAI.*T.*M.*P.*S.*G.*A.NIL),LISTLAN(*G,*S,*T,*P)
          .LISTLAN1(*G,*S,*T,*F). LSOFT(*S,*T,*U,*P,*G,*A)
          .LSOFT1(*S,*T,*U,*F,*G,*A).COB(*X.*Y.*U,*T,*TM,*NL,*NM,*NH)
          .LISTEPER(*P,*CL,*CM,*CH).INFEG(*CH,*NH).INFEC(*CM,*NM)
          . INFEC(*CL,*NL). NUMLIN(*T,*TM). MEMO(*M,*U,*T).
          .DPP(*U,*DP,*T). POSSEDE1(*A,0'.*DP). INFEG(*T,*TM)
          . FERMNBRE(*T). FERMLST(*S). FERMLST(*G)
          .FERMLST(*M).PLACEAL(*X.*Y.*U.*O,*M,*P,*A).NIL): //..
AX(TAILLE(SYS(*X.*Y.*U.*O.TAI.*T1.*M.*P.*S.*G.*A.NIL),*T),DIVEN(*T,4,*T0
          ). TRANSF(*T0,*T1). NIL): //..
AX(CONFBASE(SYS(*X.*Y.*U.*O.TAI.*T.*M.*P.*S.*G.*A.NIL),*X.*Y.*U),NIL):
          //..
AX(UNITCENT(SYS(*X.*Y.*U.*O.TAI.*T.*M.*P.*S.*G.*A.NIL),*U),NIL): //..
AX(OPTUC(SYS(*X.*Y.*U.*O.TAI.*T.*M.*P.*S.*G.*A.NIL),*O),NIL): //..
AX(*P,NIL): *P..


**  POSSEDE1(*L,*C.*M): SI *M APPARTIENT A *L AVEC CARDINALITE
              *C1, UNIFIE *C1 A *C.  SINON AJOUTE *C.*M A *L ..


POSSEDE1(*L,*C1.NIL): //..
POSSEDE1(*C1.*M1.*L,*C1.*M1): //..
POSSEDE1(*C2.*M2.*L,*C1.*M1): POSSEDE1(*L,*C1.*M1)..
```

```
**  MEMO(*M,*U,*T): CONSTRUIT LA LISTE *M DE MODULES MEMOIRE
                    POUR UNE UNITE CENTRALE *U, DE TAILLE
                    MEMOIRE *T..

MEMO(*MEM,5,*T): //. DECOMPT(*MEM,*T1,5).CONTR05(*MEM,*T1)
                .MOINX(*T1,*T2,*T). CONSMEM(*MEM,5,*T2)..
MEMO(*MEM,*UC,*T): DECOMPT(*MEM,*T1,*UC)
                .MOINX(*T1,*T2,*T).CONSMEM(*MEM,*UC,*T2)..
DECOMPT(*X,*T,*UC):VAR(*X).//..
DECOMPT(*C.*M.*MEM,*T,*UC):DEMOD(*C.*M,*TX,*UC).PLUSS(*TX,*T).
                           DECOMPT(*MEM,*T,*UC)..

DEMOD(*C.*M,*TX,*UC):MEM(*M,*TM,*UC).MULTS(*TM,*C,*TX)..
DEMOD(*C.*M,*TX,*UC):MEMX(*M,*TM,*UC).MULTS(*TM,*C,*TX)..

CONTR05(*MEM,*T):VAR(*MEM).//..
CONTR05(*MEM,*T): SOMXB5(*MEM,*NX,*NB). TRANSF(*N,*NB). INFEG(*NX,*N).
        //..
CONTR05(*MEM,*T):SOMXB5(*MEM,*NX,*NB).MOINX(*NB,*NS,*NX)
                .ADIC(*NS,SMB04,*MEM).PLUSS(*NS,*T)..
SOMXB5(*MEM,*NX,*NB):VAR(*MEM).//..
SOMXB5(*NB.SMB04.*MEM,*NX,*NB):SOMXB5(*MEM,*NX,*NB)..
SOMXB5(*C.*M.*MEM,*NX,*NB):MEMX(*M,*T,5).PLUSS(*C,*NX)
                           .SOMXB5(*MEM,*NX,*NB)..


ADIC(*C,*M,*C1.*M.*MEM):PLUSS(*C,*C1).//..
ADIC(*C,*M,*C1.*M1.*MEM):ADIC(*C,*M,*MEM)..

CONSMEM(*MEM,*UC,0):.//..

CONSMEM(*MEM,5,*T1):ADIC(*X',SMB04,*MEM).MEMX(*M,*TM,5)
                    .ADIC(*Y',*M,*MEM).MOINX(*TM',*T2,*T1)
                    .CONSMEM(*MEM,5,*T2)..
CONSMEM(*MEM,*UC,*T1):MEM(*M,*TM,*UC).ADIC(*X',*M,*MEM)
        .MOINX(*TM,*T2,*T1). CONSMEM(*MEM,*UC,*T2)..

MEMX(SMX12,0'',5):..
MEMX(SMX08,0'',5):..
MEMX(SMX04,0',5):..
MEM(SME04,0',5):..
MEM(FCM16,0'''',40):..
MEM(FCM16,0'''',65):..
MEM(SCM16,0'''',40):..
MEM(SCM16,0'''',65):..
MEM(SCM08,0'',40):..
MEM(SCM08,0'',65):..
MEM(SCS04,0',40):..
MEM(SCS04,0',65):..

MODMEM(*X): MEMX(*X,*T,*U)..
MODMEM(*X): MEM(*X,*T,*U)..
```

```
**  CONSTRUCTION DES LISTES DE LANGAGES ET DE DES SOFTWARES..

LSOFT(*S,*T,*U,*P,*G,*A): VAR(*S). //..
LSOFT(*C.*S1.*S,*T,*U,*P ,*G,*A): LOGB(*S1,*T,*U,*P1,*G1,*A1).AJL(*P1,*P)
        .AJL(*G1,*G).AJL(*A1,*A).LSOFT(*S,*T,*U,*P,*G,*A)..

LOGB(BOSA,*X1'',*U,*X'.ASR33.*Y'.FTB00.NIL,NIL,0'.ASME.0'.EDILE.
        0'.EDITEX.0'.MACP.0'.AID.NIL):..
LOGB(BOSC,*X1'',*U,*X'.ASR33.*Y'.FTB00.NIL,NIL,0'.ASME.0'.EDILE.
        0'.EDITEX.0'.MACP.0'.AID.NIL):..
LOGB(BOSD,*X'''',40,*Y'.ASR33.NIL,NIL,0'.ASMD.0'.EDILE.0'.EDITEX
        .0'.MACP.NIL): ..
LOGB(BOSD,*X'''',65,*Y'.ASR33.NIL,NIL,0'.ASMD.0'.EDILE.0'.EDITEX
        .0'.MACP.NIL): ..

LISTLAN(*X,*S,*T,*P): VAR(*X).//..
LISTLAN(0'.*G1.*G,*S,*T,*P): LAN(*G1,*S1,*T,*P1).AJL(*S1,*S)
        .AJL(*P1,*P).LISTLAN(*G,*S,*T,*P)..

LISTLAN1(*G,*S,*T,*P): LISTE(*G,LANG(*X)). ANNEXES(*G,*S,*T,*P)..

LAN(FOR16,0'.BOSD.NIL,*X'''',NIL):..
LAN(BASC,0'.BOSC.NIL,*X'''',NIL):..
LAN(APLS1,0'.BOSD.NIL,*X'''''''',NIL):..

LANG(FOR16): ..
LANG(BASC):..
LANG(APLS1):..


LSOFT1(*S,*T,*U,*P,*G,*A):LISTE(*S,LOGBA(*X)).ANNEX1(*S,*T,*U,*P,*G,*A
        )..


LOGBA(BOSA):..
LOGBA(BOSC):..
LOGBA(BOSD):..

ANNEX1(*S,*T,*U,*P,*G,*A): VAR(*S). //..
ANNEX1(*C.*S1.*S,*T,*U,*P,*G,*A): LOGB(*S1,*T,*U,*P1,*G1,*A1).
        AJL(*P1,*P). AJL(*G1,*G). AJL(*A1,*A)..

ANNEXES(*G,*S,*T,*P): VAR(*G). //..
ANNEXES(*C.*G1.*G,*S,*T,*P): LAN(*G1,*S1,*T,*P1).
        AJL(*S1,*S). AJL(*P1,*P). ANNEXES(*G,*S,*T,*P)..

**  CHOIX DU TYPE D'UNITE CENTRALE..

COB(*X.*Y.5,*W',8,16,0,0): ..
COB(*X.*Y.65,*W'''''''',256,64,0,8):..
COB(*X.*Y.40,*W'''',64,64,5,1): ..


**  CONSTRUCTION DE LA LISTE DE PERIPHERIQUES..

LISTEPER(NIL,*CL,*CM,*CH): //..
LISTEPER(*C1.*M1.*L,*CL,*CM,*CH): PERIF(*M1,*U)
        .PLUSI(*C1,*CL,*CM,*CH,*U).LISTEPER(*L,*CL,*CM,*CH)..

PLUSI(*C1,*CL,*CM,*CH,P):..
PLUSI(*C1,*CL,*CM,*CH,LDC): PLUSS(*C1,*CL)..
PLUSI(*C1,*CL,*CM,*CH,MDC): PLUSS(*C1,*CM)..
PLUSI(*C1,*CL,*CM,*CH,HDC): PLUSS(*C1,*CH)..
```

```
**  TEST DE LA PLACE ET DE L'ALIMENTATION NECESSAIRES AU SYSTEME..

PLACEAL(*X.*Y.*U.*O,*M,*P,*A):
          BAIES(*B,*Y.*Y.*U,*A). POSSEDE1(*A,0'.*B). PACKEXT(*P).
          POSSEDE1(*A,0'.*R). NUMLIM(*N,5).
          POSSEDE1(*A,*N.PWS20). FERMLST(*A).
          PLACEAL1(*X.*Y.*U.*O,*M,*P,*A). //..

PLACEAL1(*CONF,*M,*P,*A):
          ENCPARTIEL(*CONF,*C1C,*C2C,*AC,*UC,*VC,*PC,*MC).
          ENCAL(*M,*C1C,*C1M,*C2C,*C2M,*AC,*AM,*UC,*UM,*VC,*VM,*PC,*PM,*
          MC,*MM). ENCAL(*P,*C1M,*C1P,*C2M,*C2P,*AM,*AP,*UM,*UP,*VM,*VP,
          *PM,*PP,*MM,*MP). ENCAL(*A,*C1P,*C1A,*C2P,*C2A,*AP,*AA,*UP,*UA
          ,*VP,*VA,*PP,*PA,*MP,*MA). PT(*C1A,*C2A,*AA,*UA,*MA,*PA,*VA)..

PT(*C1A,*C2A,*AA,*UA,*MA,*PA,*VA):
          SUP(*UA,0). SUP(*MA,0). SUP(*PA,0).
          PLUZ(*MA,*PA,*Z). MULTZ(*Z,24,*Z1). DIVA(*Z1,10,*Z2).
          PLUZ(*Z2,*VA,*VT). SUP(*VT,0)..

TEST(*C1,*C2): SUP(*C1,0). MULT(*C1,2,*CR). PLUZ(*CR,*C2,*CT).
          SUP(*CT,0)..


PACKEXT(NIL):..
PACKEXT(PSX06):..

BAIES(NIL,*X.*T.*Y,*A):..
BAIES(BAB14,*Y.B.*Y,*A):..
BAIES(BAX36,*X.B.*Y,*A): BAIES(*Z,*X.B.*Y,*A).POSSEDE1(*A,0'.*Z)..

ENCAL(NIL,*C1,*C1,*C2,*C2,*A,*A,*U,*U,*V,*V,*P,*P,*M,*M):..
ENCAL(*C1.*M1.*L,*C1E,*C1S,*C2E,*C2S,*AE,*AS,*UE,*US,*VE,*VS,
          *PE,*PS,*ME,*MS): TRANSF(*E1,*C1). OPER(*M1,*E1,*MC1,*MC2,*MA,
          *MU,*MV,*MP,*MM). PLUZ(*MC1,*C1E,*C1I). PLUZ(*MP,*PE,*PI).PLUZ
          (*MC2,*C2E,*C2I). PLUZ(*MA,*AE,*AI). PLUZ(*MU,*UE,*UI). PLUZ(*
          MV,*VE,*VI). PLUZ(*MM,*ME,*MI).
          ENCAL(*L,*C1I,*C1S,*C2I,*C2S,*AI,*AS,*UI,*US,*VI,*VS,*PI,*PS,*
          MI,*MS)..

OPER(*M1,*E1,*MC1,*MC2,*MA,*MU,*MV,*MP,*MM): ENCPARTIEL(*M1,*NC1,*NC
          2,*NA,*NU,*NV,*NP,*NM). MULTZ(*E1,*NC1,*MC1). MULTZ(*E1,*NC2,*
          MC2).MULTZ(*E1,*NA,*MA). MULTZ(*E1,*NU,*MU). MULTZ(*E1,*NV,*MV
          ). MULTZ(*E1,*NP,*MP). MULTZ(*E1,*NM,*MM)..
```

```
**  LISTE(*L,*P): CONSTRUIT DANS *L UNE LISTE D'INDIVIDUS
                  SATISFAISANT LA PROPRIETE *P..

LISTE(*L,*P): ..
LISTE(*L,*P): UNIV(*P,*N.*X.NIL). ONA(*P). NOT(AP(*X,*L)).
          POSSEDE1(*L,Ø'.*X). UNIV(*P1,*N.*Y.NIL). LISTE(*L,*P1)..

ONA(*P): *P..

AP(*X,NIL): //. IMPASSE..
AP(*X,*C.*X.*L): //..
AP(*X,*C.*Y.*L): AP(*X,*L)..

FERMLST(NIL)://..
FERMLST(*C1.*M1.*L): FERMNBRE(*C1).//.FERMLST(*L)..
FERMLST(*L1.*L):FERMLST(*L1).//.FERMLST(*L)..
FERMLST(*L):..

POSSEDE(SYS(*X.*Y.*U.*O.TAI.*T.*M.*P.*S.*G.*A.NIL),*C1.*M1):
          HAS(*M.*P.*S.*G.*A,*C1.*M1). //..
POSSEDE(*L,*C1.*M1): POSSEDE1(*L,*C1.*M1)..

HAS(*M.*P.*S.*G.*A,*C1.*M1): MODNEM(*M1).//.POSSEDE1(*M,*C1.*M1)..
HAS(*M.*P.*S.*G.*A,*C1.*M1): LOGD(*M1,*X,*Y,*Z,*W,*V). //
           .POSSEDE1(*S,*C1.*M1)..
HAS(*M.*P.*S.*G.*A,*C1.*M1): PERIF(*M1,*X).//.POSSEDE1(*P,*C1.*M1)..
HAS(*M.*P.*S.*G.*A,*C1.*M1): LAN(*M1,*X,*Y,*Z).//.POSSEDE1(*G,*C1.*M1)..
HAS(*M.*P.*S.*G.*A,*C1.*M1): POSSEDE1(*A,*C1.*M1)..

AJL(NIL,*L):..
AJL(*C.*M.*L1,*L): POSSEDE1(*L,*C.*M).AJL(*L1,*L)..
```

```
**  OPERATIONS SUR LES CARDINAUX DE LA FORME :  *X'''''' '..

TRANSF(*N,*X): VAR(*N). //. TRADG(*N,*X)..
TRANSF(Ø,*X): VAR(*X).//..
TRANSF(Ø,Ø): //..
TRANSF(*N,*X'): INF(Ø,*N).MOINS(*N,1,*M). TRANSF(*M,*X)..

TRADG(Ø,*X): VAR(*X). //..
TRADG(Ø,Ø): ..
TRADG(*N,*X'): TRADG(*M,*X). PLUS(1,*M,*N)..


**  FERMNBRE(*X): REMPLACE LA VARIABLE DU CARDINAL *X PAR Ø ..

FERMNBRE(Ø): //..
FERMNBRE(*X'): FERMNBRE(*X)..

** PLUSS(*Y,*X): UNIFIE *X AVEC LE RESULTAT DE L'ADDITION DE *X ET *Y..


PLUSS(*Y,*X): VAR(*X). SSULP(*Y,*X). //..
PLUSS(*Y,*X'): PLUSS(*Y,*X)..

SSULP(Ø,*X): //..
SSULP(*Y,*X): VAR(*Y). //..
SSULP(*Y',*X'): SSULP(*Y,*X)..


** MULTS(*C1,*C2,*C): UNIFIE *C AVEC LE PRODUIT DE *C1 PAR *C2..



MULTS(Ø,*T2,*T): ..
MULTS(*X',*T2,*T): PLUSS(*T2,*T). MULTS(*X,*T2,*T)..

MOINX(*X,*Y,*Y):VAR(*X).//..
MOINX(Ø,*Y,*Y)://..
MOINX(*X,*Y,Ø)://.ECHEC..
MOINX(*X,*Y,*Z):VAR(*Z)..
MOINX(*X',*Y,*Z'):MOINX(*X,*Y,*Z)..

NUMLIM(Ø,*L): ..
NUMLIM(*N',*L): INF(Ø,*L). MOINS(*L,1,*L1). NUMLIM(*N,*L1)..

NUMER(Ø):..
NUMER(*C'): NUMER(*C)..
NUMER(*C'): NUMER(*C)..

INFEG(*X,*N): TRANSF(*X1,*X). SUP(*N,*X1)..
```

```
**  OPERATIONS ARITHMETIQUES AVEC LES ENTIERS..

MULTZ(0,*B,0): //..
MULTZ(*A,0,0): //..
MULTZ(-*A,-*B,*C): //. MULT(*A,*B,*C)..
MULTZ(-*A,*B,-*C): //. MULT(*A,*B,*C)..
MULTZ(*A,-*B,-*C): //. MULT(*A,*B,*C)..
MULTZ(*A,*B,*C): MULT(*A,*B,*C)..

PLUZ(-*A,-*B,-*C): //. PLUS(*A,*B,*C)..
PLUZ(-*A,*B,-*C): INF(*B,*A). //. MOINS(*A,*B,*C)..
PLUZ(-*A,*B,*C): //. MOINS(*B,*A,*C)..
PLUZ(*A,-*B,*C): //. PLUZ(-*B,*A,*C)..
PLUZ(*A,*B,*C): PLUS(*A,*B,*C)..

DIVEX(0,*DIV,0):..
DIVEX(*D,*DIV,*R): MOINS(*D,1,*D1). DIV(*D1,*DIV,*R1).
          PLUS(*R1,1,*R)..

SUP(*A,*A): //..
SUP(-*A,-*B): //. INF(*A,*B)..
SUP(*A,-*B): //..
SUP(-*A,*B): //. IMPASSE..
SUP(*A,*B): INF(*B,*A)..

DIVA(*X,0,*X): //. LIGNE. SORM("DIV PAR ZERO"). LIGNE..
DIVA(-*X,-*Z,-*W): //. DIV(*X,*Z,*W)..
DIVA(-*X,*Z,0): DIV(*X,*Z,*W). EG(*W,0). //..
DIVA(-*X,*Z,-*W): //. DIV(*X,*Z,*W)..
DIVA(*X,-*Z,0): DIV(*X,*Z,*W). EG(*W,0). //..
DIVA(*X,-*Z,-*W): //. DIV(*X,*Z,*W)..
DIVA(*X,*Z,*W): DIV(*X,*Z,*W)..
```

** PPIX1(*X,*Y): DEFINIT LE PRIX D'UN MODULE *X COMME *Y,
EN MILLIERS DE NOUVEAUX FRANCS..

```
PPIX1(SMX04,5): //..
PPIX1(SMB04,6): //..
PPIX1(SMX12,13): //..
PPIX1(FCM16,30)://..
PPIX1(SCM16,22)://..
PPIX1(SCM08,15)://..
PPIX1(SCS04,26)://..
PPIX1(BAB36,5)://..
PPIX1(BAX36,4)://..
PPIX1(DRP40,8)://..
PPIX1(DRP65,16)://..
PPIX1(ASR33,11)://..
PPIX1(FTB00,20)://..
PPIX1(FOR16,2)://..
PPIX1(BASC,4)://..
PPIX1(APLS1,10)://..
PPIX1(PWS20,3): //..
PPIX1(SMX08,9)://..
PPIX1(PTE00,5)://..
PPIX1(MHB05,57)://..
PPIX1(BC.T.5.0,22)://..
PPIX1(BC.T.5.1,22)://..
PPIX1(EC.T.5.2,23)://..
PPIX1(EC.T.5.0,26)://..
PPIX1(EC.T.5.1,27)://..
PPIX1(EC.T.5.2,27)://..
PPIX1(BC.D.5.0,24)://..
PPIX1(EC.D.5.1,25)://..
PPIX1(BC.D.5.2,25)://..
PPIX1(BC.D.5.3,25)://..
PPIX1(BC.D.5.4,27)://..
PPIX1(MTS05,2)://..
PPIX1(MPD00,6)://..
PPIX1(EC.T.40.0,51)://..
PPIX1(BC.T.40.1,52)://..
PPIX1(EC.T.40.0,57)://..
PPIX1(EC.T.40.1,58)://..
PPIX1(BC.D.40.0,54)://..
PPIX1(BC.D.40.1,55)://..
PPIX1(BC.D.40.2,58)://..
PPIX1(EC.D.40.0,60)://..
PPIX1(EC.D.40.1,61)://..
PPIX1(EC.D.40.2,63)://..
PPIX1(BC.B.40.0,57)://..
PPIX1(EC.B.40.1,58)://..
PPIX1(EC.B.40.0,63)://..
PPIX1(EC.B.40.1,64)://..
PPIX1(EC.B.65.0,124)://..
PPIX1(EC.B.65.1,125)://..
PPIX1(BAB14,2)://..
PPIX1(RSK06,9)://..
PPIX1(FOR16,2)://..
PPIX1(BOSD,2)://..
PPIX1(*X,0):..



PERIF(MLP01,LDC):..
PERIF(PTE00,P):..
PERIF(ASR33,P): ..
PERIF(FTB00,LDC):..
PERIF(MHB05,HDC):..
```

```
ENCPARTIEL(BC.T.5.0,4,0,0,0,140,15,20):/..
ENCPARTIEL(BC.T.5.2,4,0,0,0,140,15,20):/..
ENCPARTIEL(BC.T.5.1,5,01,1,0,140,15,20):/..
ENCPARTIEL(EC.T.5.0,2,0,2,0,140,15,20):/..
ENCPARTIEL(EC.T.5.2,2,0,2,0,140,15,20):/..
ENCPARTIEL(EC.T.5.1,6,1,1,0,140,15,20):/..
ENCPARTIEL(BC.D.5.0,4,0,0,9,140,15,20):/..
ENCPARTIEL(BC.D.5.2,4,0,0,9,140,15,20):/..
ENCPARTIEL(BC.D.5.3,4,0,0,9,140,15,20):/..
ENCPARTIEL(BC.D.5.1,6,1,1,7,140,15,20):/..
ENCPARTIEL(BC.D.5.4,4,0,0,23,140,15,20):/..
ENCPARTIEL(EC.D.5.0,2,0,0,9,140,15,20):/..
ENCPARTIEL(EC.D.5.2,2,0,0,9,140,15,20):/..
ENCPARTIEL(EC.D.5.3,2,0,0,9,140,15,20):/..
ENCPARTIEL(EC.D.5.1,4,1,1,9,140,15,20):/..
ENCPARTIEL(EC.D.5.4,2,0,0,23,140,15,20):/..
ENCPARTIEL(BC.B.5.0,0,12,1,28,140,15,20):/..
ENCPARTIEL(BC.B.5.1,0,12,1,23,140,15,20):/..
ENCPARTIEL(BC.B.5.2,4,0,0,31,140,15,20):/..
ENCPARTIEL(EC.B.5.3,6,1,0,29,140,15,20):/..
ENCPARTIEL(EC.B.5.0,0,10,1,23,140,15,20):/..
ENCPARTIEL(EC.B.5.1,0,10,1,28,140,15,20):/..
ENCPARTIEL(EC.B.5.2,2,0,0,31,140,15,20):/..
ENCPARTIEL(EC.B.5.3,5,1,1,29,140,15,20):/..
ENCPARTIEL(RSX06,0,13,1,-6,170,20,20):/..
ENCPARTIEL(BAB14,0,0,0,14,0,0,0):/..
ENCPARTIEL(BAB36,0,0,0,36,0,0,0):/..
ENCPARTIEL(BAX36,0,0,0,36,0,0,0):/..
ENCPARTIEL(DRP40,0,-1,0,0,3,0,0):/..
ENCPARTIEL(DRP65,0,-1,0,0,3,0,0):/..
ENCPARTIEL(PVS20,0,0,-1,0,200,20,20):/..
ENCPARTIEL(SMB04,0,-1,0,0,-17,-5,0):/..
ENCPARTIEL(SMX04,0,-1,0,0,0,0,0):/..
ENCPARTIEL(SMX08,0,-1,0,0,0,0,0):/..
ENCPARTIEL(SMX12,0,-1,0,0,0,0,0):/..
ENCPARTIEL(SCH08,-1,0,0,0,-22,-5,-3):/..
ENCPARTIEL(SCM16,-1,0,0,0,-22,-5,-3):/..
ENCPARTIEL(SCS04,-1,0,0,0,-33,-9,-3):/..
ENCPARTIEL(BC.T.40.0,5,1,1,0,100,15,17): /..
ENCPARTIEL(BC.T.40.1,5,1,1,0,100,15,17): /..
ENCPARTIEL(EC.T.40.0,4,1,1,0,98,15,17): /..
ENCPARTIEL(EC.T.40.1,4,1,1,0,98,15,17): /..
ENCPARTIEL(BC.D.40.0,5,1,1,7,100,15,17): /..
ENCPARTIEL(BC.D.40.1,5,1,1,7,100,15,17): /..
ENCPARTIEL(BC.D.40.2,5,1,1,21,100,15,17): /..
ENCPARTIEL(EC.D.40.0,4,1,1,7,98,15,17): /..
ENCPARTIEL(EC.D.40.1,4,1,1,7,98,15,17): /..
ENCPARTIEL(EC.D.40.2,4,1,1,21,98,15,17): /..
ENCPARTIEL(BC.B.40.0,12,1,3,24,100,15,17): /..
ENCPARTIEL(BC.B.40.1,12,1,3,24,100,15,17): /..
ENCPARTIEL(EC.B.40.1,11,1,3,24,100,15,17): /..
ENCPARTIEL(EC.B.40.0,11,1,3,24,100,15,17): /..
ENCPARTIEL(EC.B.65.0,9,1,3,24,25,11,15): /..
ENCPARTIEL(FCM16,0,-2,0,0,0,0,0):/..
ENCPARTIEL(FTB00,0,-1,0,-4,-15,0,0):/..
ENCPARTIEL(*X,0,0,0,0,0,0,0):..


DRP(40,DRP40,*T): TRANSF(*T1,*T). SUP(*T1,9). //..
DRP(65,DRP65,*T): TRANSF(*T1,*T). SUP(*T1,9)./..
DRP(*X,NIL,*T):..
```

```
** SORTIE EN CLAIR DU SYSTEME COMPOSE..

SORTIR(SYS(*X.*Y.*U.*O.TAI.*T.*M.*P.*S.*G.*A.NIL)): //. LIGNE. LIGNE.
          SORM("CARACTERISTIQUES DU SYSTEME DEMANDE: "). LIGNE.
          SORM("=====================================". LIGNE. LIGNE.
          SORM("- CONFIGURATION DE BASE: ").SORT(*X.*Y.*U).
          SORM(", OPTION: "). SORT(*O). LIGNE.
          SORM("- TAILLE DE LA MEMOIRE CENTRALE: "). TRANSF(*TI,*T).
          MULT(*TI,4,*TA). SORT(*TA). SORM(" K MOTS"). LIGNE.
          SOP2(*M.*P.*S.*G.*A)..
SORTIR(NIL): //..
SORTIR(*V): SORLISTE(*V). LIGNE. LIGNE..

SOP2(*M.*P.*S.*G.*A): SORM("- STRUCTURE DES MODULES DE MEMOIRE: ").
          LIGNE.SORTIR(*M).LIGNE. SORM("- LISTE DE PERIPHERIQUES :").
          LIGNE. SORTIR(*P).LIGNE. SORM("- SOFTWARE :"). LIGNE.
          SORTIR(*S). LIGNE. SORM("- LANGAGES :"). LIGNE. SORTIR(*G).
          LIGNE. SORM("- MODULES OPTIONNELS :"). LIGNE. SORTIR(*A)..

SORLISTE(0.*M.*X): //. SORLISTE(*X)..
SORLISTE(NIL)://..
SORLISTE(*C.*M.*X): //. LIGNE. TRANSF(*C1,*C).SORT(*C1). SORM("  ").
          SORT(*M). SORLISTE(*X)..
SORLISTE(*C): SORT(*C)..

SORP(*X): LIGNE. SORM("- PRIX DU SYSTEME :")
          . SORT(*X). SORM("000 FRANCS")..

SORTAX(*P): SORT(*P). LIGNE..
```

INDEX