HAL
open science

Etude critique et données de compilation du langage Cobol

Jean-Loup Baer

To cite this version:

Jean-Loup Baer. Etude critique et données de compilation du langage Cobol. Génie logiciel [cs.SE]. Université Joseph-Fourier - Grenoble I, 1963. Français. NNT : . tel-00164736

HAL Id: tel-00164736

https://theses.hal.science/tel-00164736v1

Submitted on 23 Jul 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre :

THÈSE

présentée à

LA FACULTÉ DES SCIENCES DE L'UNIVERSITÉ DE GRENOBLE

pour obtenir

LE TITRE DE DOCTEUR DE TROISIÈME CYCLE

Mathématiques Appliquées

---

par

Jean-Loup BAER

Ingénieur I. E. G., I. M. A. G.

Licencié ès Sciences

ÉTUDE CRITIQUE ET DONNÉES DE COMPILATION DU LANGAGE COBOL

Thèse soutenue le 27 Juin 1963 devant la Commission d'examen :

Monsieur J. KUNTZMANN, Président

Messieurs B. VAUQUOIS, Examinateurs

N. GASTINEL

L. BOLLIET

FACULTE DES SCIENCES DE L'UNIVERSITE DE GRENOBLE

DOYENS HONORAIRES M. FORTRAT P.
M. MORET L. Membre de l'Institut

DOYEN / WEIL L.

PROFESSEURS :

MM.	WOLFERS F.	PHYSIQUE
	NEEL L.	PHYSIQUE EXPERIMENTALE Membre de l'Institut
	TORIER A.	ZOOLOGIE
	HEILMANN R.	CIMIE ORGANIQUE
	KRAVTCHENKO J	MECANIQUE RATIONNELLE
	CHABAUTY C.	CALCUL DIFFERENTIEL ET INTEGRAL
	PRADE M.	POTAMOLOGIE
	BENOIT J.	RADIOELECTRICITE
	CHENE M.	CHIMIE PAPETIERE
	BESSON J.	ELECTROCHIMIE
	WEIL L.	THERMODYNAMIQUE
	FELICI N.	ELECTROSTATIQUE
	KUNTZMANN J.	MATHEMATIQUES APPLIQUEES
	BARBIER R	GEOLOGIE APPLIQUEE
	SANTON L	MECANIQUE DES FLUIDES
	OZENDA P.	BOTANIQUE
	FALLOT M.	PHYSIQUE INDUSTRIELLE
	MOUSSA A.	CHIMIE NUCLEAIRE
	TRAYNARD P.	CHIMIE
	SOUTIF M	PHYSIQUE
	CRAYA A.	HYDRODYNAMIQUE
	REB G.	MATHEMATIQUES M.P.C
	BLAMBERT M	MATHEMATIQUES
	BONNIER E	ELECTROCHIMIE
	DESSAUX G.	PHYSIOLOGIE ANIMALE
	PILLET E	ELECTROTECHNIQUE
	DEBELMAS J.	GEOLOGIE
	VAUQUOIS B.	MATHEMATIQUES APPLIQUEES
	PEAUD-LE-NOEL	BIOSYNTHESE DE LA CELLULOSE
	GALVANI O.	MATHEMATIQUES
	REULOS R.	THEORIE DES CHAMPS
	AYANT	PHYSIQUE APPROFONDIE
	GALLISSOT F.	MATHEMATIQUES APPLIQUEES
Melle	LUTZ E.	MATHEMATIQUES
	BOUCHEZ R.	PHYSIQUE NUCLEAIRE
	LLIBOUTRY L.	GEOPHYSIQUE
	MICHEL R.	GEOLOGIE ET MINERALOGIE
	GERBER R.	MATHEMATIQUES
	PAUTHENET R.	ELECTROTECHNIQUE

PROFESSEURS SANS CHAIRE :

MM.	SILBER R	MECANIQUE DES FLUIDES
	MOUSSIEGT J.	ELECTRONIQUE
	BARBIER J.C	PHYSIQUE
	BUYLE-BODIN	ELECTRONIQUE
Mme	KOFLER	BOTANIQUE
	DREYFUS	THERMODYNAMIQUE
	VAILLANT	ZOOLOGIE ET HYDROBIOLOGIE
	GIRAUD P.	GEOLOGIE
	SAVELLI M.	PHYSIQUE GENERALE

PROFESSEURS ASSOCIES :

MM.	REIZNICK	PHYSIOLOGIE VEGETALE
	LUMER	MATHEMATIQUES

MAITRES DE CONFERENCES :

Mme	LUMER L.	MATHEMATIQUES
	PERRET R.	SERVOMECANISMES
	ARNAUD P.	CHIMIE
Mme	BERBIER M.J.	ELECTROCHIMIE
	BRISSONNEAU	PHYSIQUE
	COHEN J.	ELECTROTECHNIQUE
Mme	SOUTIF J.	PHYSIQUE
	DEPASSEL R.	MECANIQUE
	ROBERT	CHIMIE PAPETIERE
	ANGLES D'AURIAC	MECANIQUE DES FLUIDES
	BIAREZ J.	MECANIQUE PHYSIQUE
	COUMES A.	ELECTRONIQUE
	DODU J.	MECANIQUE DES FLUIDES
	DUCROS P.	MINERALOGIE ET CRISTALLOGRAPHIE
	GIDON P.	GEOLOGIE ET MINERALOGIE
	GLENAT R.	CHIMIE
	HACQUES G.	CALCUL NUMERIQUE
	LANCIA R.	PHYSIQUE AUTOMATIQUE
	PEBAY-PEROULA J.	PHYSIQUE
	GASTINEL N.	MATHEMATIQUES APPLIQUEES
	LACAZE A.	THERMODYNAMIQUE
	GAGNAIRE D.	CHIMIE PAPETIERE
	DEGRANGE D.	ZOOLOGIE
	KLEIN J.	MATHEMATIQUES
Mme	KAHANE J.	PHYSIQUE
	RASSAT	CHIMIE SYSTEMATIQUE
	DEPORTES C.	CHIMIE
	DEPOMMIER P.	PHYSIQUE NUCLEAIRE
	POLOUJADOFF M.	ELECTROTECHNIQUE
	BARRA J.	MATHEMATIQUES APPLIQUEES
Mme	BOUCHE L.	MATHEMATIQUES
	PERRIAUX J.	GOELOGIE
	SARROT-REYNAUD	GEOLOGIE
	CAUQUIS G.	CHIMIE GENERALE
	LABRE A.	BOTANIQUE
	BETHOUX P.	MATHEMATIQUES APPLIQUEES
	BONNET G.	PHYSIQUE GENERALE

Je tiens à exprimer ma profonde reconnaissance à :

Monsieur le Professeur KUNTZMANN, Directeur du Laboratoire de Calcul, qui a dirigé mon travail, m'a donné de précieux conseils et a bien voulu me faire l'honneur de présider le Jury.

Monsieur le Professeur VAUQUOIS pour l'intérêt qu'il a bien voulu apporter à cette étude.

Monsieur GASTINEL, Maitre de conférences, qui a accepté d'être membre du Jury.

Monsieur BOLLIET, Ingénieur au C.N.R.S., qui, par sa connaissance des langages de programmation et ses méthodes de compilation, a su orienter la dernière partie de ce travail.

°°°

INTRODUCTION

La prolifération des langages symboliques de programmation est devenue excessive. Certains s'en sont émus et ont tentés de réaliser des langages orientés universels. Pour l'analyse numérique et plus généralement pour les applications scientifiques, l'ALGOL (algorithmic language) semble être adopté par beaucoup d'utilisateurs. Pour la gestion administrative et comptable COBOL (COmmon Business Oriented language) a vu le jour en 1960. Remanié et amélioré en 1961, COBOL commence à être utilisé aux Etats-Unis.

Dès son apparition COBOL a soulevé de vives critiques souvent justifiées mais quelque fois également dues à la méconnaissance des buts que s'étaient assignés les auteurs du rapport original. Depuis, le second rapport a rendu caduques certaines des critiques et un nouveau rapport est en préparation, mais il n'en reste pas moins qu'à l'heure actuelle, aucun compilateur existant (cf Appendice 2) ne peut souffrir la comparaison, au point du vue performance, avec les autocodeurs écrits pour des machines spécifiques.

Mais COBOL ne se veut pas uniquement un langage de programmation mais encore un langage d'analyse c'est-à-dire d'approche cohérente de la machine, écrit avec les mots de tous les jours, suivant une syntaxe proche de celle de la langue courante et un langage permettant la compatibilité entre différentes machines, compatibilité qui n'a pas l'espoir de se voir totale car comme nous le verrons ceci est impossible, mais maximum.

Que ces buts ne puissent être réalisés simultanément et qu'ils entraînent de graves défauts est vrai, mais que COBOL présente un certain nombre d'avantages sur les autocodeurs commerciaux le précédant ne l'est pas moins.

Le but de cette étude est de :

1. Présenter rapidement COBOL (en Français);
2. Le comparer aux langages commerciaux qui l'ont pré-

cédé et en faire la critique du point de vue langage commercial.

3. Le comparer à ALGOL et le critiquer en tant que langage peu formel.

4. Présenter les grandes lignes d'une compilation sur une machine binaire à mots avec toutes les difficultés que cela présente.

5. Conclure sur l'avenir de COBOL.

°°°

I. PRESENTATION RAPIDE DE COBOL

1. STRUCTURE GENERALE DE COBOL.

1. 1. Divisions

Ce programme écrit en COBOL, suivant la traduction donnée par l'E C M A |2|, est divisé en quatre parties qui doivent apparaître dans l'ordre suivant :

a) IDENTIFICATION : dont le but est d'identifier le programme source et les états fournis par le compilateur pour chaque programme.

b) EQUIPEMENT : qui spécifie le "milieu extérieur" c'est-à-dire la machine et le type de matériel nécessaire aussi bien pour la compilation que pour l'exécution du programme généré, et la déclaration des fichiers.

c) DONNEES : décrivant le "milieu interne" et la structure des informations c'est-à-dire donnant un modèle détaillé des fichiers, articles, mémoires de manoeuvre et constantes utilisées et traitées par le programme.

d) TRAITEMENT : dans laquelle se trouve les instructions opérationnelles et de gestion que devra exécuter la machine et dont la succession constitue le "traitement" de l'information.

1. 2. Jeu de caractères

Les caractères utilisés dans les mots du langage, autres que les libellés sont tirés des groupes A à Z , 0 à 9 et le trait d'union.

Un mot contient au plus 30 de ces caractères et ne peut comprendre d'espaces (les libellés sont une exception) cf. I 3a

Les autres caractères utilisés sont :

. Pour la ponctuation " , ; . ( ) nous symboliserons l'espace par □)

. Pour les formules = + - x (multiplié par et **
puissance) / (divisé-par)
. Pour les relations = < >
. Pour l'édition $ ¤ (certification de chèque) , .
Soit 51 caractères si l'on matérialise l'espace.

Un mot se termine par un . , ;) suivi d'un espace ou par un ou plusieurs espaces. Un mot ne peut commencer ou se terminer par un -

Les libellés constituent des exceptions à cette règle.

1. 3. Types de mots :

On distingue les types de mots suivants : noms, verbes et mots réservés.

a) Parmi les noms on distingue :

. Les "noms de donnée" contenant au moins un caractère alphabétique et qui désigne une donnée spécifiée dans les divisions DONNEES ou EQUIPEMENT.

Ex. : QUARANTE-EN-STOCK PRIX A-97

. Les "noms de condition" qui sont des noms donnés à une (ou plusieurs) valeur que peut prendre un nom de donnée.

. Les "étiquettes" qui désignent des paragraphes ou des sections.

. Les "libellés" dont la valeur est égale à celle des caractères les composant.

On distingue les "libellés numériques" composés uniquement de caractères 0 à 9 , - , + et . (virgule décimale),

Ex. : 0123.456/-99

Les libellés non numériques qui doivent se trouver entre guillemets "FAUX MOUVEMENT" et les constantes figuratives qui sont des libellés auxquels on a donné des noms fixes. Ce sont ZERO, ESPACE, GUILLEMETS, TOUS.
BORNE INF, BORNE SUP, MINIMUM, MAXIMUM.

. Un registre spécial dénommé COMPTEUR de 5 chiffres décimaux.

. Les "noms spéciaux" destinés à désigner certains dispositifs technologiques de la machine utilisée.

b) Les verbes :

Ce sont des mots uniques, apparaissant dans la division traitement et qui désignent une action spécifique - MULTIPLIER, LIRE, EXECUTER.

c) Les mots réservés sont des mots utilisés dans des buts syntaxiques et ne peuvent servir de noms ou de verbes. On distingue :

. Les conjonctions DANS notifiant une qualification (cf I. 4. 2.) et ET qui peut remplacer la "," dans les clichés.

. Les mots-facultatifs qui facilitent la lecture du langage et qui n'affectent en rien la syntaxe. Ex. LORSQUE, QUAND.

. Les mots clés qui doivent être utilisés conformément aux normes des différents clichés (of Appendice) LONGUEUR, CONSTANTES, PAR, EN, SUPERIEUR, ENTREE, BOBINE.

1. 4. Ponctuation

Une virgule ou un point-virgule peuvent apparaître, pour faciliter la lecture du programme, aux endroits où ils sont indiqués dans les clichés. Leur emploi est facultatif.

Lorsqu'un point est indiqué dans un cliché, il doit figurer pour que la syntaxe soit correcte.

2. DIVISIONS DONNEES.

2. 1. Structure de la division DONNEES.

L'information à traiter peut être de trois types :

. Les données, au sens usuel du mot et les résultats appartenant respectivement au fichier entrée, et au fichier sortie.

. Les résultats intermédiaires des calculs qui sont

placés dans des mémoires de manoeuvre.

. Les constantes

A ces trois types de données correspondent les sections FICHIERS, INTERMEDIAIRES et CONSTANTES qui sont formées d'une succession d'indications comportant un numéro de niveau, un nom de donnée, et une série de clauses.

2. 2. La section FICHIERS :

Quand on définit une information à partir d'un fichier, on doit distinguer les aspects physiques du fichier (par ex : les repères de début et fin de fichiers, les groupements d'articles par blocs sur le support extérieur etc...) auxquels correspond la description de fichiers, et les définitions explicites de chaque entité logique du fichier auxquelles correspondent les descriptions d'articles.

2. 3. La description de Fichier :

A chaque fichier correspond la rubrique de description de fichier qui a pour N° de niveau DF , pour nom de donnée un nom de fichier qui aura dû apparaître précédemment dans la division EQUIPEMENT, (cf. I. 3. 4.) et une série de clauses dont les suivantes sont obligatoires :

. Une spécification des REPERES de début et de fin de FICHIER qui doit figurer même si ces repères n'existent pas sur le fichier traité (option REPERES OMIS).

. Les NOMS DES ARTICLES du fichier.

. Le nombre d'articles ou de caractères par bloc (sauf s'il n'y a qu'un article/bloc).

De plus des renseignements supplémentaires sur le nombre d'articles dans le fichier, de caractères par article , sur le mode d'enregistrement externe, sur les indicatifs de tri et sur les identificateurs de repères peuvent être ajoutés suivant la structure du fichier.

(Pour le format général de la description de fichier cf Appendice)

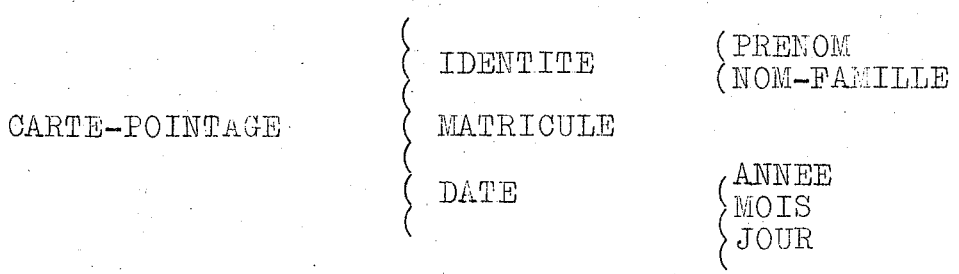
2. 4. La description d'article :

2. 4. 1. Le concept de niveau :

Avant d'aborder les différentes clauses qui peuvent apparaître à la suite d'un nom de donnée, il est préférable de préciser ce que l'on appelle article.

PAR article, on considère un terme générique qui désigne un groupe logique d'informations se rapportant au même sujet. Par exemple DATE est le nom générique des informations ANNEE, MOIS, JOUR. Ainsi DATE ne représente pas un simple élément d'information, mais le groupe d'éléments d'information (que nous appellerons "mots") qui compose cet article.

Pour mentionner les données d'un article, on est amené à donner des niveaux à chaque groupe représentant une subdivision de l'article. Ainsi si sur une CARTE-DE-POINTAGE on doit indiquer l'IDENTITE de l'employé, son MATRICULE, la DATE etc..., mais dans l'IDENTITE on doit indiquer le PRENOM et le NOM-DE-FAMILLE et dans la DATE, l'ANNEE, le MOIS, le JOUR, on subdivisera suivant les niveaux suivants :



Pour préciser la structure d'un article, on est donc amené à définir :

- L'article courant qui est l'article lui-même niveau Ø1
- Le mot-groupe qui est un ensemble de mots-élémentaires niveau Ø 2 à 48
- Le mot élémentaire qui ne peut être subdivisé niveau Ø1 à 49.

Le numéro de niveau d'un mot (groupe ou élémentaire)

suivant le dernier mot élémentaire d'un groupe doit être celui d'un des mots groupe auquel le dernier mot appartient. Dans l'exemple, ceci veut dire que MATRICULE doit avoir le même numéro que IDENTITE.

2. 4. 2. La description d'article courant :

La rubrique de description d'article courant se compose donc du numéro de niveau Ø1, du nom de l'article courant qui a dû être déjà déclaré lors de la description de fichier correspondante (cf I. 2. 3) et des clauses optionnelles donnant la LONGUEUR et la CLASSE de cet article (id à mot-groupe cf I. 2. 4. 3.)

2. 4. 3. La description de mot-groupe :

Elle se compose d'un numéro de niveau compris entre Ø2 et 48, du nom de donnée et des clauses suivantes selon les cas.

. Si le mot-groupe se trouve plusieurs fois successives sur le fichier, par ex : 4 , il n'est pas nécessaire de donner 4 fois sa description. Il suffit d'indiquer qu'il FIGURE 4 FOIS.

(On peut également décrire le cas de répétions dont le nombre est sujet à certaine condition). Dans le traitement, ce mot sera indicié.

. On peut définir la CLASSE de mot-groupe, c'est-à-dire s'il est ALPHABETIQUE, NUMERIQUE ou ALPHANUMERIQUE. Tous les mots élémentaires se rapportant à ce mot groupe auront la même classe. De même on peut définir son USAGE dominant c'est-à-dire sa forme INTERNE OU EXTERNE de représentation.

. A titre de vérification, on peut définir sa LONGUEUR qui devra être égale à la somme des longueurs des mots élémentaires, qui le composent, la longueur étant définie en termes de caractères.

2. 4. 4. La description de mot-élémentaire :

Elle se compose d'un numéro de niveau de Ø2 à 49, du nom de donnée ou du mot-clé INDEFINI et d'une série de clauses à choisir suivant les cas, mais dont deux sont obligatoires.

. La LONGUEUR ou les limites de la longueur.

La CLASSE et l'USAGE sont s'ils ont été déclarés pour le mot-groupe et dans ce cas, ces options peuvent compléter celles du mot-groupe. Ainsi NUMERIQUE n'est pas en contradiction avec ALPHANUMERIQUE.

Lorqu'à la place du nom de donnée, on emploie le mot INDEFINI, cela signifie que l'on veut identifier une information, qui est effectivement présente dans l'article, mais qui n'a pas à être mentionnée dans le programme.

Comme pour les mots groupes, si le mot élémentaire apparaît successivement plusieurs fois dans le mot groupe, on l'indique en nommant le nombre de fois où il FIGURE.

Mais lorsqu'on est au niveau du mot élémentaire, il faut donner le modèle détaillé du mot. On peut donc spécifier s'il est ALGEBRIQUE, l'emplacement de la VIRGULE (à GAUCHE ou à DROITE) s'il est CADRE ou PLACE dans un mot machine d'une manière non - standard, l'ETENDUE des valeurs qu'il peut prendre et enfin des clauses d'édition telles que par ex. : ZEROS SUPPRIMES, CERTIFIER, SYMBOLE FLOTTANT.

Certaines de ces options peuvent être écrites de manière plus abrégée à l'aide de MODELE qui permettent également d'indiquer des insertions de caractères tels le . (virgule décimale réelle) les signes CR ou DB , + ou - etc...

2. 5. Les noms de condition :

Nous avons vu (I. 1. 4.) la définition des noms de condition. Dans une rubrique de description d'article, on les identifie par un numéro de niveau spécial : 88. L'intérêt de ces noms de condition est de condenser le programme lors de la division TRAITEMENT. Ainsi, si l'on affecte à un nom de donnée, par exemple VILLE les noms de condition :

```
88   PARIS  ;  VALEUR  1  et  3  JUSQU'A  6
88   LYON   ;  VALEUR  7 .
88   MARSEILLE :  VALEUR  8  .
```

Au lieu d'avoir à écrire dans la division TRAITEMENT (cf I. 4. 12) : ST VILLE EGALE 1 OU (SUPERIEUR à 2 ET INFERIEUR à 7)il suffira d'écrire :

ST PARIS

2. 6. La clause redéfinit

Lorsqu'à l'intérieur d'un article le programmeur veut faire partager la même zone de mémoire, lors du traitement, à deux mots il écrit que le mot - 2 REDEFINIT le mot - 1 , celui-ci ayant été décrit juste avant le mot - 2 .

Lorsqu'on utilise cette option, elle doit être obligatoirement la première de la description d'article du mot 2. Toutes les autres clauses peuvent être données ensuite dans un ordre quelconque.

2. 7. L'option copier

Si le programmeur doit définir deux articles ou deux mots de structure exactement semblable (par exemple dans une remise à jour de fichier , les articles du fichier entrée et du fichier sortie auront la même structure logique), il n'a pas à réécrire toute la description d'article mais il suffit d'indiquer au compilateur qu'il doit COPIER une certaine description d'article .

2. 8. La clause recouvre

Cette clause donne la possibilité de grouper les mots élémentaires d'une deuxième façon, avec chevauchement éventuel. La description d'articles d'un mot qui RECOUVRE d'autres mots doit commencer avec le n° de niveau 66.

2. 9. La section intermédiaires

Cette section décrit les mémoires de manoeuvre (ou intermédiaires) que réclame le programme. Ces mémoires seront accessibles pendant toute l'exécution du programme généré;

Une description d'intermédiaire est semblable à une description d'article. On distingue les intermédiaires isolées des intermédiaires groupées.

Les intermédiaires isolées, c'est-à-dire celles dont les positions relatives en machine n'ont pas d'importance ont un n° de niveau égal à 77. Elles doivent comporter les clauses CLASSE et LONGUEUR (ou la clause MODELE les remplaçant).

Les intermédiaires groupées sont assimilées à des articles et doivent commencer par une description d'article courant. Toutes les options de la description d'article sont applicables aux intermédiaires. Des noms de condition peuvent s'appliquer aux intermédiaires groupés.

Lors de l'introduction initiale du programme généré, chaque intermédiaire est garnie de la VALEUR donnée par cette option. Si cette option ne figure pas, la VALEUR n'est pas définie.

2. 10. La section CONSTANTES.

Les rubriques CONSTANTES sont rédigées de la même façon que pour la section INTERMEDIAIRES. On distingue encore les constantes isolées, à n° de niveau 77 qui doivent posséder les clauses CLASSE, LONGUEUR (ou MODELE) et la clause VALEUR.

Les constantes groupées, servent en général à définir des tables. On ne peut y faire figurer des noms de condition et l'option VALEUR doit figurer au niveau élémentaire. Lorsqu'on veut employer des indices pour utiliser cette table, celle-ci doit être REDEFINIE pour montrer la structure hiérarchique qui lui est inhérante.

3. LES DIVISIONS IDENTIFICATION ET EQUIPEMENT.

3. 1. La Division identification :

Elle est placée en tête du programme et ne comporte qu'un paragraphe obligatoire, celui donnant l'identification du programme. D'autres renseignements facultatifs, tels l'auteur,

les dates d'écriture et de compilation, des remarques, peuvent y être ajoutés.

3. 2. La division équipement.

La fonction de base de cette division est d'organiser et spécifier les aspects de l'ensemble du programme (aussi bien source que généré), qui dépendent des caractéristiques particulières de la ou des machines employées. En outre, elle donne la description du milieu extérieur de tous les fichiers utilisés.

Elle comprend deux sections, CONFIGURATION et ENTREE-SORTIE.

3. 3. La section configuration.

Les trois paragraphes composant cette section ont les buts suivants : les deux premiers obligatoires appelés COMPILATION et EXECUTION décrivent respectivement les machines utilisées pour compiler le langage source et exécuter le programme généré, la dimension de mémoires qui est affecté à ce travail, le nombre d'éléments technologiques nécessaire à chaque partie. Le troisième facultatif, nommé NOMS-SPECIAUX (cf I. 1. 3) a pour but de désigner, par des noms mnémoniques, des éléments technologiques particuliers (boutons allumés ou éteints, clé "en" ou "hors") qui seront testés de même façon que les noms de condition (cf I. 1. 5).

3. 4. La section entrée-sortie.

Elle dst divisée en deux paragraphes, GESTION-FICHIERS et GESTION-E-S. LE premier affecte un milieu externe à chaque fichier, donnant en plus les aspects particuliers de ce fichier tels que la possibilité de plusieurs bobines, son éventualité de présence, la réservation de zone tampon lors de l'exécution du programme généré. Les renseignements concernant les techniques spéciales d'entrée-sortie, les reprises et les affectations de zone de mémoires communes sont donnés dans le deuxième.

4. LA DIVISION TRAITEMENT

4. 1. Structure.

La division traitement contient les instructions, rédigées en un langage proche du langage courant, qui sont nécessaires à la réalisation du programme. Cette division est scindée en SECTIONS, elles-mêmes comportant un ou plusieurs paragraphes. Les sections et les paragraphes sont nommés par des étiquettes (cf I. 1. 3.). Un paragraphe est composé d'une ou plusieurs phrases. Une phrase est composée d'une ou plusieurs instructions, la dernière se terminant par un point suivi d'un espace.

On distingue trois sortes d'instructions :

- Les instructions impératives ;
- Les instructions conditionnelles ;
- Les instructions de déclaration.

4. 2. Qualification :

Avant d'aborder l'étude des différents genres d'instruction, nous devons en terminer avec tout ce qui a rapport à ce que l'on peut appeler les identificateurs, c'est-à-dire les noms de donnée, noms de condition et étiquettes. Tout nom auquel on se réferre dans un programme COBOL doit être unique. On peut le réaliser simplement en donnant à chaque nom une orthographe particulière Mais nous avons vu dans l'option COPIER (cf I. 2. 7.) qu'il était souvent pratique de donner un même nom à des mots appartenant à des articles différents. Pour rendre ce nom unique, on le "qualifiera" au moyen de noms à hiérarchie inférieure, sans laisser d'ambiguité, ceci au moyen d'un nombre convenable de DANS suivi d'un nom à hiérarchie inférieure. De la même façon, il est possible de "qualifier" des noms de paragraphe, le qualificatif étant le nom de section à laquelle appartient le paragraphe.

4. 3. Indices :

Lorsqu'on a décrit un mot à l'aide de la clause FIGURE, ce mot ne pourra être cité qu'en faisant suivre son nom d'un indice, qui sera soit un libellé numérique entier, soit un nom de donnée à valeur entière.

Dans les deux cas, l'indice suit le mot auquel il se rapporte et écrit entre parenthèses. On peut aller jusqu'à 3 niveau d'indices, correspondant à des descriptions de tables à trois dimensions. Dans ce cas, les indices sont tous entre les mêmes parenthèses et séparés par des virgules.

Il est à noter qu'un nom de données à qualifier ne peut-être indicié alors qu'un qualificatif de hiérarchie la plus inférieure peut l'être. Ainsi :

C DANS B DANS A (5, 4, 2) est correct.

et

C (2) DANS B (4) DANS A (5) est incorrect.

4. 4. Les instructions impératives et conditionnelles :

Une instruction impérative se compose soit d'un verbe et de ses opérandes, soit d'une suite d'instructions impératives. Une suite d'instructions impératives peut comprendre une instruction ALLER (cf I. 4. 8.) ou une instruction ARRET FINAL (cf I. 4. 9.) qui, si présente, doit figurer comme dernière instruction impérative de la suite où elle figure.

Une instruction conditionnelle peut-être de forme :

. Forme 1 SI condition Instruction 1 SINON Instruction 2

. Forme 2 Instruction impérative suivie d'une instruction conditionnelle.

Dans le cas (2), l'instruction impérative ne peut se terminer par une instruction ALLER ou ARRET. Les instructions comprenant un verbe arithmétique avec l'option DEPASSEMENT (cf I. 4. 6) où le verbe LIRE avec l'option FIN (cf. I. 4. 7.) sont considérées du 2ème type.

Dans le cas (1), l'instruction 1 et/ou l'instruction 2 peuvent être impératives ou conditionnelles et, dans ce dernier cas, le processus peut se rejeter. L'instruction 1 ou l'instruction 2 peut ne pas figurer et être remplacée par les deux mots-clés PHRASE SUIVANTE qui font un branchement sur la phrase suivante. Dans le cas où le groupement SINON PHRASE SUIVANTE est

immédiatement suivi d'un point, il peut être omis et ce processus se reporte sur le groupement précédent.

Au point de vue ponctuation, on peut se servir des séparateurs " ; " et ALORS entre deux instructions dans les cas d'instructions impératives ou dans le cas d'instructions conditionnelles entre la condition et l'instruction 1 et entre l'instruction 1 et SINON.

4. 5. Les verbes :

A la base d'une instruction, on trouve le verbe. COBOL en autorise vingt-trois qui sont :

. 5 verbes arithmétiques :	AJOUTER (ou ADDITIONNER) SOUSTRAIRE MULTIPLIER DIVISER CALCULER
. 6 verbes d'entrée-sortie :	OUVRIR FERMER LIRE ECRIRE RECEVOIR INDIQUER
. 3 verbes de branchement :	ALLER EXECUTER CHANGER
. 2 verbes de transfert de données :	TRANSFERER EXAMINER
. Le verbe d'arrêt :	ARRET
. 4 verbes de gestion du compilateur :	LANGAGE CONTINUER NOTE INCLURE

. 2 verbes de déclaration : UTILISER
DEFINIR
(INCLURE pouvant également faire partie de ce groupe).

Les formats de ces verbes sont donnés en appendice.

4. 6. Les verbes arithmétiques :

Les quatre premiers verbes arithmétiques : AJOUTER, SOUSTRAIRE, MULTIPLIER et DIVISER ((1)) correspondent aux quatre opérations élémentaires. Dans le cas d'AJOUTER, le nombre d'opérandes est quelconque ainsi que le nombre d'opérandes qu'on peut SOUSTRAIRE d'un nom de donnée.

Si l'on veut que le résultat de l'opération soit transféré en un nom de donnée spécifique, il faut utiliser la clause RESULTAT suivie de ce nom de donnée. Si ceci n'est pas fait, le résultat sera transféré automatiquement dans le dernier opérande. Ces opérations peuvent être ARRONDIES. Si l'on veut faire un traitement spécial en cas de dépassement de capacité, on indiquera ce traitement après le mot clé DEPASSEMENT situé à la fin de l'instruction. Si cette clause n'est pas indiquée et que le dépassement de capacité survient, les résultats sont imprévisibles.

Le dernier verbe arithmétique, CALCULER, autorise l'emploi de formules. Dans ces formules, les opérateurs ont la hiérarchie suivante :

le plus haut	* *	ou PUISSANCE
puis	*	ou MULTIPLIE-PAR ou FOIS et/ou DIVISE-PAR
puis	+	ou PLUS et - ou MOINS

((1)) Par suite du cliché anglais DIVIDE.... INTO.... on autorise les deux clichés français : DIVISER-PAR....QUANTITE..
et : DIVISER.... PAR....

avec bien entendu des hiérarchies supérieures pour les parenthèses.

Ainsi CALCULER $X = A + B/C^{*H} + D ** E * F - G$

sera interprété comme: CALCULER $X = A + (\frac{B}{C}) * H + (D^{E}) * F - G$

Les clauses d'ARRONDI et de DEPASSEMENT sont également utilisables pour le verbe CALCULER.

4. 7. Les verbes d'entrée-sortie :

Avant toute opération sur un fichier, on devra l'OUVRIR, ce qui traitera tout ce qui est relatif aux repères et le positionnera pour délivrer ou recevoir le premier article. Dans le cas d'un fichier entrée, on devra LIRE le fichier et les traitements en cas de fin-de-fichier seront décrits par l'instruction suivant la clause FIN-DE-FICHIER qui est obligatoire après une instruction LIRE par fichier. Dans le cas où elle n'est pas répétée, on considère que c'est la même pour chaque opération LIRE du fichier. Dans le cas d'un fichier-sortie, on devra ECRIRE les articles avec, dans le cas de l'imprimante, des possibilités de SAUT D'INTERLIGNES. (A remarquer qu'on "lit" les fichiers et qu'on "écrit" les articles).

Enfin, pour terminer le traitement des bobines et des fichiers entrée et sortie, on doit FERMER ces fichiers, avec possibilité ou non de REBOBINAGE ou VERROUILLAGE.

On peut introduire des informations de faible volume dans les machines, c'est-à-dire les RECEVOIR et ce DEPUIS certains supports d'entrée. De même, on peut les INDIQUER "en clair" SUR d'autres supports.

4. 8. Les verbes de branchement :

Lorsqu'on veut faire une rupture de séquence, on écrit ALLER A l'étiquette où l'on veut continuer le traitement. Si l'on a différents choix d'étiquettes suivant la valeur entière d'un nom de donnée, on écrit cette liste d'étiquettes après ALLER A, liste que l'on fait suivre du mot clé SELON suivi lui-même du nom de donnée dont la valeur n déterminera la n° étiquette du branchement.

Le forçage des aiguillages se fait par l'intermédiaire de l'instruction CHANGER. Si un paragraphe ne contient qu'une instruction ALLER, on modifie la séquence prédéterminée en indiquant CHANGER le nom du paragraphe POUR PASSER A une certaine étiquette.

L'exécution des boucles de programmes, portant sur un ou plusieurs paragraphes, sont décrites par l'instruction EXECUTER. On peut indiquer le nombre de FOIS exactes où l'on veut faire la boucle, ou la LIMITE de ce nombre par la réalisation d'une condition. Cette LIMITE peut d'ailleurs être fonction d'indices (maximum 3) dont on donnera les valeurs de départ, les pas de progression, les LIMITES respectives et leur ordre de succession.

4. 9. Les verbes de transfert :

Les transferts d'une zone émettrice en une ou plusieurs zones réceptrices s'effectuent par le verbe TRANSFERER. Ces transferts s'effectuent suivant les modèles des zones réceptrices, pour tout ce qui concerne l'édition, suppression de caractères, "rembourrage" de zéros ou d'espaces. En fait, chaque utilisateur en définissant les différents USAGE des mots qu'il utilise devra en même temps prévoir les règles concernant l'instruction TRANSFERER.

Lorsqu'on veut remplacer un caractère dans un mot par un autre caractère, on doit EXAMINER ce mot REMPLACANT tel caractère ou telle suite de caractères PAR un autre. Utilisant le registre spécial COMPTEUR, on peut également EXAMINER en COMPTANT ces caractères puis en les REMPLACANT PAR un autre.

4. 10. Le verbe d'arrêt :

Comme son nom l'indique, l'instruction ARRET sert à provoquer soit un arrêt momentané pendant le traitement, soit l'ARRET FINAL lorsque le programme est achevé.

4. 11. Le verbe de gestion du compilateur :

Il est possible d'incorporer au programme généré des programmes écrits en langage de la machine utilisé pour l'exploi-

tation. Ceci se fait en écrivant LANGAGE suivi du nom du langage de programmation. Quand on veut revenir au langage COBOL, on écrit LANGAGE COBOL. Lorsqu'on veut utiliser des sous-programmes existant sous forme COBOL en bibliothèque, il faut INCLURE ces programmes, avec possibilité de remplacement de paramètres par des noms de données spécifiques au programme que l'on est en train de traiter.

Des commentaires peuvent être introduits par NOTE suivi de ces commentaires qui ne seront pas pris en charge par le compilateur.

Enfin, lorsqu'on utilise les verbes EXECUTER et UTILISER, il est nécessaire, lorsqu'il y a plusieurs voies possibles, de fournir un point de sortie commun pour un sous-programme. Ceci est fait en écrivant un paragraphe composé uniquement de son étiquette et du verbe CONTINUER. Dans le traitement normal en séquence, le compilateur ne prendra pas cette instruction en charge.

4. 12. Les conditions :

Nous avons vu qu'une instruction conditionnelle devait commencer par le mot clé SI qu'on peut rattacher à un verbe puisqu'il implique une action, ce mot clé étant suivi d'une condition. Cette condition peut-être simple ou composée.

Une condition simple est formée de l'un des quatre types de tests suivants :

- Test de relation ;
- Test de classe ;
- Test de variable conditionnelle ;
- Test d'état d'interrupteur.

a) Les tests de relation. Ce sont des comparaisons entre deux opérandes (noms de donnée, libellés ou formules) portant sur leur égalité, supériorité ou infériorité. Pour deux opérandes numériques, les mots EGAL, SUPERIEUR,

INFERIEUR ou NON suivi de ces trois mots, DIFFERENT, DEPASSE, ont des significations évidentes. Pour des opérandes alphanumériques, on convient que c'est la valeur du caractère qui importe, le mot le plus court en cas d'inégalité de longueur étant complété par des espaces.

Les tests de relation peuvent également être à un opérande, l'autre opérande étant implicitement zéro. On a alors les tests POSITIF, NEGATIF, ZERO ou NON suivi de l'un de ces trois mots.

b) Test de classe. Dans le cas de mots alphanumériques, on peut tester la classe de ce mot, c'est-à-dire s'il est NUMERIQUE ou ALPHABETIQUE.

c) Test de variable conditionnelle. Ce sont les tests portant sur les noms de condition (cf. I. 2. 5.)

d) Test d'états d'interrupteur. L'interrupteur déclaré comme un des NOMS-SPECIAUX... joue le rôle d'un nom de condition.

Des conditions simples peuvent être combinées avec les opérateurs logiques ET, OU, NON formant alors une condition composée. La seule restriction concerne le NON qui doit précéder une parenthèse ouvrante ou une condition simple ne comportant pas de NON. S'il n'y a pas de partenthèse le ET a prédominance sur le OU.

4. 13. La section déclaration :

Si, dans le traitement, on veut se servir d'instructions de déclaration, ces instructions doivent être placées en tête du programme précédées du mot-clé DECLARATION . Lorsqu'elles sont toutes écrites, on indique BORNE DECLARATIONS. Chacune des instructions de déclaration est une section unique. L'instruction UTILISER sert à préciser les traitements qui doivent compléter les traitements des repères normaux assurés par la gestion entrée-sortie. L'instruction DEFINIR introduit de nouveaux verbes dont le cliché est décrit à l'aide de paramètres formels.

5. CARACTERISTIQUES PARTICULIERES.

5. 1. Le format de référence :

La feuille de programmation COBOL comprend quatre parties :

a) Les colonnes 1-6 qui servent à numéroter les lignes séquentiellement. Ce numéro n'est pas obligatoire et n'a aucun effet sur le programme.

b) La colonne 7. Lorsqu'un mot n'est pas terminé sur la ligne précédente un trait d'union en colonne 7 indique qu'il se continue. En cas de blanc en colonne 7, la ligne précédente s'est terminée par un mot complet.

c) Les colonnes 8-12. Les noms de division, de section, de paragraphes doivent commencer en colonne 8. Dans la division DONNEES, les DF doivent également commencer en colonne 8. Les entrées suivantes peuvent également commencer en col. 8 ou être décalées de quatre colonnes pour montrer l'organisation hiérarchique des données.

d) Les colonnes 12 à (généralement) 72 servent pour écrire le texte COBOL.

5. 2. La bibliothèque :

Des sous-programmes appartenant aux sections équipement, données ou traitement peuvent faire partie d'une bibliothèque de sous-programmes COBOL. Pour les appeler, il suffit pour les divisions EQUIPEMENT et dONNEES d'écrire le nom de paragraphe ou le nom de donnée suivi de COPIER le nom du sous-programme cherché. Dans la division TRAITEMENT, on emploie le verbe INCLURE (cf. 1 4. 1[1]).

5. 3. Segmentation :

Les sections de la division TRAITEMENT peuvent posseder un numéro de priorité. Les sections les plus utilisées au-

ront les numéros de priorité les plus faibles.

Les sections ayant le numéro 0 au numéro spécifié dans la division EQUIPEMENT après SEGMENTER (cf. Appendice), ou 49 si cette clause n'existe pas, sont inclus dans la partie fixe du programme généré.

Les sections depuis le numéro spécifié après SEGMENTER jusqu'à 49 seront inclues dans le programme généré tant qu'il y aura de la place disponible. Les sections qui ne pourraient y figurer seront maintenues comme des segments individuels composés de sections groupées par ordre de priorité.

Les sections de numéro 50 à 89 seront considérées comme segments individuels. Elles seront automatiquement réinitialisées chaque fois qu'elles seront introduites.

Les sections de numéro 89 à 99 ne peuvent être introduites qu'une fois dans le programme généré et ne peuvent être rappelées ensuite.

°°°

II. ETUDE CRITIQUE DE COBOL.

1. INTRODUCTION.

Les buts primordiaux de COBOL, tels que compatibilité, langage de programmation et système d'analyse, langage écrit avec des mots employés couramment et suivant une syntaxe très proche de la langue utilisée quotidiennement vont se montrer irréalisables simultanément et provoquer des longueurs, redondances, etc... De plus, COBOL comme beaucoup d'autres langages symboliques, contient des ambiguités dues souvent à l'imprécision de la définition de certaines options.

Pourtant COBOL présente un progrès sensible par rapport aux langages commerciaux qui l'ont précédé, ou qui ont été écrits au même moment. Notre comparaison portera sur FLOW-MATIC (UNIVAC) [3] COMTRAN (IBM Commercial Translator) [4], FACT (HONEYWELL) [5], NEBULA (FERRANTI) [6].

Chacun de ces autocodes était destiné à un type de machine déterminé répondant à des standards définis par les constructeurs. On ne pouvait donc parler de compatibilité.

2. LA COMPATIBILITE DANS COBOL.

2. 1. Divisions EQUIPEMENT et DONNEES.

Au sens strict, on pourrait dire qu'un langage est compatible, si étant donné un programme source, on peut le compiler sur des machines différentes sans lui apporter aucune modification. La seule notion de division EQUIPEMENT rend impossible une telle définition pour COBOL. Mais la restriction ne s'arrête pas seulement à cette division. Dans la description des données, un programme n'aura pas la même structure s'il s'agit de fichiers qui seront traités par une machine binaire ou une machine décimale, une machine à caractères ou une machine à mots. Ainsi dans un programme COBOL écrit pour un 1401 ou GAMMA-30, un mot élémentaire numérique devant subir des traitements arithmétiques

pourra être décrit par :

TARTEMPION ; LONGUEUR 4 ; NUMERIQUE.

Ce même mot, utilisé dans un fichier appelé à être traité par un 7044 ou 7090 sera - en supposant une programmation orientée vers l'efficacité - décrit comme

TARTEMPION ; LONGUEUR 4 ; NUMERIQUE INTERNE PLACE DROITE.

ce qui signifie que le mot est en binaire pur, cadré à droite dans un mot machine entier qui lui est réservé.

Enfin pour le GAMMA-60, on écrirait :

TARTEMPION ; LONGUEUR 4 ; NUMERIQUE INTERNE-1.

ce qui signifie que le mot est en flottant sur deux catènes.

La description de base LONGUEUR 4 ; NUMERIQUE est évidemment la même pour les différentes machines. Mais dans les deux derniers cas, l'absence d'options supplémentaires créerait soit des erreurs, soit une organisation irrationnelle demandant un travail supplémentaire au compilateur lui faisant perdre ainsi une grande efficacité.

Cet exemple n'est pas unique. Le nombre d'USAGE d'une donnée est défini par l'utilisateur, et il semble normal que chaque utilisateur définisse les USAGE lui convenant le mieux. Mais en plus de cette notion ambigüe, vient s'ajouter celle de PLACE DROITE et PLACE GAUCHE, qui décrit la place de la donnée dans le mot-machine et qui pourra encore donner différentes interprétations, par exemple dans la cohabitation de mots représentés par des codes différents dans un même mot machine. D'autre part, les traitements et descriptions de REPERES seront différents pour chaque compagnie tant que ceux-ci ne seront pas standardisés. Enfin les jeux de caractères, perforation et impression, n'étant pas les mêmes, ce qui a trait aux clauses d'édition et à l'insertion de caractères dans les MODELES est sujet à caution.

Pour la compatibilité de la division DONNEES, on peut dire

que l'analyse générale du programme, le découpage logique des articles, les options essentielles sont compatibles c'est-à-dire que la compatibilité est respectée pour la forme écrite ou ce que l'on pourrait appeler le format externe, mais que des retouches devront être apportées dès que l'on veut décrire le format interne.

2. 2. Division TRAITEMENT.

Pour la division TRAITEMENT, on peut parler de compatibilité quasi-complète, mais il est nécessaire de prendre certaines précautions. Tout d'abord, certains verbes tels RECEVOIR et INDIQUER dépendent de la technologie de la machine et par exemple le 7044 ne peut RECEVOIR aucune donnée. Il en est de même lorsqu'on fait référence à des NOMS-SPECIAUX. Pour le verbe ENTRER, dont la définition est confuse, il peut-être interprété comme signifiant que l'on va exécuter un sous-programme écrit dans un langage compris par la machine spécifique pour laquelle le programme source est écrit. On ne peut dire que ceci soit compatible puisque cette séquence devra être réécrite pour chaque machine.

Mais il y a également d'autres cas où la compatibilité peut-être remise en cause |7|. Supposons que dans le langage source, on ait l'instruction suivante :

SI CARACTERE<A OU>9 ALORS ALLER A P1 SINON ALLER P2.

et que α, β, γ soient des caractères spéciaux autorisés par COBOL.

Il peut se présenter deux machines dont l'une ait pour séquence de caractères :

β γ A B C .... Z α 0 I ... 9

et l'autre :

β γ 0 I ... 9 α A B C .... Z

S'il se trouve que la valeur de CARACTERE soit α, pour la première machine il y aura rupture de séquence vers P2, pour

la deuxième vers P1.

Cette incompatibilité est levée si dans la division DONNEES on affecte à CARACTERE un nom de condition, par exemple :

88 TYPE ; VALEUR α .

et si dans la division TRAITEMENT on écrit :

SI TYPE ALORS ALLER A P1 SINON ALLER A P2.

Cependant, en supposant la compatibilité dans les instructions, il n'est pas certain que les résultats soient identiques, après calculs sur une machine décimale ou sur une machine binaire. En effet les règles COBOL pour les opérations arithmétiques déclarent que les résultats intermédiaires pourront dépasser d'un digit à gauche ou à droite la longueur du plus grand opérande. Ce digit n'aura pas la même valeur suivant la machine utilisée et le test DEPASSEMENT aura alors une signification différente.

En conclusion, on peut dire que, mis à part les programmes d'essai très simples le degré de compatibilité est le suivant:

. Aucun programme COBOL ne peut être transféré automatiquement sans :

1°. Ecrire de nouveau la division EQUIPEMENT.

2°. S'assurer que les données sont décrites pour le même type de machine et dans le cas contraire modifier tout ce qui se rapporte à la configuration interne de ces données.

3°/ S'assurer que ce qui se rapporte dans la division TRAITEMENT aux aspects technologiques de la machine-1 existe également sur la machine-2.

. En général la division TRAITEMENT ne sera pas remaniée et la description logique de la division DONNEES restera la même, c'est-à-dire l'analyse du problème est compatible.

x x x

3. COBOL OUTIL D'ANALYSE ET DE PROGRAMMATION.

En définissant COBOL, ses auteurs ont voulu, non seu-

lement en faire un nouveau langage de programmation, mais de plus faire en sorte qu'en programmant en COBOL, l'utilisateur ait une approche cohérente de la machine. Ils ont cherché à donner un enchainement logique aux divisions et sections qui se succèdent. En fait, on peut considérer un programme écrit en COBOL comme la solution théorique d'un problème décrit avec les étapes naturelles suivantes :

a) Le titre et éventuellement son domaine d'utilisation.

b) Les outils dont on aura besoin pour réaliser la solution pratique, et les caractéristiques physiques des matériaux utilisés dans le problème. C'est ce qui constitue la division EQUIPEMENT et ce que l'on pourrait comparer à la description des appareils de mesure et des constituants d'un problème d'analyse chimique par exemple.

c) La description des données et des résultats qu'on veut obtenir ce qui, en poursuivant notre analogie, pourrait être la description des caractéristiques particulières à chaque constituant (couleur, acidité...).

d) Le traitement des données pour arriver aux résultats, ceci par exemple

Pour revenir à un programme commercial, une fois que sont décrites les données et qu'un organigramme synthétique a été établi, il n'y a aucune difficulté à écrire un programme en COBOL. Prenons un exemple simple : une remise à jour de fichier. Tout d'abord, le programmeur analyse l'équipement dont il a besoin. Si le problème comporte trois fichiers, un fichier entrée qui est le fichier à mettre à jour, un fichier entrée qui comporte les mouvements et un fichier sortie qui est le fichier mis à jour, il affectera par l'option

PRENDRE... , METTRE SUR... .

chaque fichier à un élément périphérique, en prévoyant les bascules, reprises, puis les zones tampons, les zones communes aux fichiers entrée et sortie à mettre et mis à jour puisqu'ils ont la même structure.

Ensuite dans la division DONNEES il définira cette

structure. Le fichier mis à jour sera composé d'articles divisés en mots, d'une part alphanumérique pour ce qui est repérage de chaque produit, d'autre part numérique pour ce qui est montant, quantité, prix... se rapportant à chaque produit. Le programmeur, s'il veut utiliser correctement COBOL devra à ce moment là employer les options les plus favorables selon le type de machine utilisée pour limiter les temps d'exploitation et les conversions éventuelles de données de forme mémorisation en forme traitement.

Puis il écrira le programme proprement dit et ceci d'une façon très simple. Par exemple, un test sur des indicatifs pourra s'écrire :

SI NUMERO DANS PERMANENT INFERIEUR A NUMERO DANS MOUVEMENT ALORS ALLER A ECRIRURE SINON SI SUPERIEUR ALLER A ERRONNE.

Il n'y a alors plus de difficulté non seulement à écrire le programme mais de plus la compréhension du programme est aisée même pour quelqu'un n'ayant pas l'habitude de la programmation.

Mais ceci présente le double inconvénient d'alourdir considérablement le langage source et c'est ce que nous allons voir au paragraphe suivant, et de demander une compilation qui est d'autant plus longue que le langage est éloigné du langage machine.

x^x^x

4. LONGUEURS, REDONDANCES, IMPRECISIONS.

Comme nous venons de le voir, il est facile de lire et d'écrire un programme COBOL. Pourtant cette écriture est fastidieuse et comporte de nombreuses redondances. D'autre part, l'usage de mots qui dans certains cas sont facultatifs, dans d'autres obligatoires peut créer des confusions, et l'emploi de synonymes est superflu ceci aussi bien en Anglais que dans la traduction française.

4. 1. Divisions EQUIPEMENT et DONNEES.

Dans la division EQUIPEMENT ces inconvénients apparaissent peu. Pourtant lorsqu'on déclare un fichier, on doit écrire :

PRENDRE fichier, METTRE SUR élément.

Il serait plus simple d'écrire :

METTRE FICHIER-ENTREE fichier SUR élément.

ce qui éliminerait le verbe PRENDRE qui est inutile, et introduirait déjà la notion de fichier-entrée ou sortie ce qui permettrait une codification plus rapide comme nous le verrons plus tard.

C'est dans la division DONNEES qu'apparaissent les redondances les plus nombreuses. Au lieu de ne décrire que les mots élémentaires d'une façon complète, on peut décrire tous les mots à quelque niveau qu'ils soient étant donné bien entendu que les mots élémentaires eux doivent être décrits complètement et que certaines clauses tels le signe, emplacement de la virgule, symboles d'édition etc... leur sont réservés. Mais si nous prenons la clause LONGUEUR nous pouvons l'écrire à tous les niveaux, ce qui peut paraître une vérification mais ce qui ést également cause d'erreurs de programmation. D'autre part, cette information de longueur devra être nécessairement calculée par le compilateur, afin de donner des adresses à chaque mot. Comme elle est donnée en termes de caractères, cela ne correspondra à rien pour une machine à mots si on a imposé des conditions spéciales de cadrage afin d'obtenir un programme généré plus efficace (clauses USAGE, PLACE, CADRE). On aurait eu plutôt intérêt à imposer la règle suivante : seules les LONGUEURS de l'article courant et des mots élémentaires devront être données , ce qui d'un côté supprime beaucoup de redondances permises, et de l'autre permet une vérification et laisse jouer un rôle spécial à l'article de niveau 01.

La seule clause que l'on peut considérer comme clause à maintien est celle définissant la CLASSE des mots. Mais un mot groupe ALPHANUMERIQUE, pouvant être composé de mots élémentaires ALPHABETIQUE et NUMERIQUE, soit encore ALPHANUMERIQUE, ce main-

tien est encore souvent sujet à précision. Il faudra donc tester la CLASSE (et également l'USAGE) des mots élémentaires.

L'utilisation de MODELE se justifie en COBOL par ce que cette option apporte de nouveau par rapport aux autres clauses : l'insertion de symboles d'édition qu'il n'est pas possible de représenter autrement. Il serait possible de décrire tous les mots élémentaires de la division DONNEES sous forme de MODELE (sauf les clauses FIGURE, de positionnements spéciaux et de valeurs) mais cela serait aller à l'encontre de la lisibilité du programme.

La qualification qui peut aussi bien apparaître dans la division DONNEES que dans la division TRAITEMENT est l'exemple type de longueur inutile. Il serait beaucoup plus rapide d'imposer, dans la division EQUIPEMENT par exemple, un suffixe de caractère différent pour chaque fichier, et dans la division DONNEES un nouveau suffixe pour chaque mot groupe d'un article courant ayant des mots élémentaires nécessitant une qualification supplémentaire. Ceci ferait gagner non seulement du temps à la programmation mais encore dans la compilation à la recherche de la codification inhérente au mot qualifié. On pourra objecter qu'étant donné qu'il y a 49 niveaux possibles de qualification, on pourrait avoir à introduire 49 suffixes, et de ce fait dépasser la limite autorisée de 30 caractères par mot. Mais d'une part il paraît improbable qu'un programmeur puisse jamais faciliter la programmation en donnant le même nom à deux hiérarchies de 49 niveaux (ce qui est le but de la qualification) car à chaque référence du mot de niveau 49, il lui faudrait écrire 49 fois DANS suivi d'un nom de donnée et d'autre part le nombre de 49 niveaux est véritablement superflu et il est difficile d'imaginer des problèmes où les articles se décomposent en plus de 10 niveaux. Il serait plus simple de se fixer une dizaine de niveaux correspondant chacun à une décomposition précise comme article, sous-article, mot groupe principal etc... jusqu'au mot élémentaire.

Si la qualification paraît disproportionnée par rapport au but cherché, par contre les noms de condition sont une économie

de programmation. Cette économie n'apparaît qu'au niveau du langage source, car au moment de la compilation on sera forcé de reprendre la forme originale afin de générer une variable booléenne correspondant au nom de condition.

4. 2. Division TRAITEMENT.

Dans la division TRAITEMENT, le premier problème qui se pose est celui relatif à la ponctuation et aux séparateurs d'instructions. On sait que le seul signe de ponctuation imposé est le point à la fin d'une phrase. Mais il est possible d'insérer un point-virgule ou le mot ALORS entre 2 instructions quelqu'en soit leur nature. Ainsi il est permis d'écrire :

AJOUTER X A Y ; MULTIPLIER Y PAR Z ALORS DIVISER Z PAR T RESULTAT W.
SI C < D SI E < F ; AJOUTER X Y MULTIPLIER Z PAR T; SINON ALLER A P1.

Dans la première phrase, le point-virgule et ALORS ou leur absence éventuelle ont peu d'importance immédiate mais dans la seconde les séparateurs doivent jouer le rôle de parenthèses et la structure véritable est la suivante :

SI C < D ALORS (SI E < F ALORS (AJOUTER X, Y ; MULTIPLIER Y PAR T) SINON ALLER A P1) SINON PHRASE SUIVANTE.

Il faudra donc lors d'une phase de la compilation préalable à la génération introduire des séparateurs auxiliaires formalisés qui permettront de générer des étiquettes de programme lors de la formation du programme objet. Ceci peut-être défini de la façon suivante :

. Une phrase se termine toujours par un point.

. Dans une instruction impérative (non incluse dans une instructio n conditionnelle) les instructions opérationnelles, c'est-à-dire le verbe et ses opérandes, seront séparées par des points-virgules.

. Une instruction conditionnelle aura la forme suivante,

au niveau post-codification :

SI cond ALORS instr1 SINON instr2.

Si l'instr1 et (ou) l'instr2 est une instruction impérative, les instructions opérationnelles seront séparées par un code AUSSI. Nous obtiendrons pour l'exemple précédent :

SI C < D ALORS SI E < F ALORS AJOUTER X, Y AUSSI MULTIPLIER Y PAR T SINON ALLER A P1.
Le point faisant office de SINON PHRASE SUIVANTE pour le premier SI.

Au moment de la codification, il faudra également donner des codes différents aux points finissant les phrases et ceux délimitant une étiquette, ceci étant rendu aisé par le format de référence qui impose aux étiquettes de débuter en colonne 8. de la feuille de programme.

La virgule en COBOL ne joue aucun rôle syntaxique. Elle permet facultativement, de séparer les composants d'une liste (opérande, indices, étiquettes etc...). Son emploi au niveau du langage source facilitera la compilation car elle renseignera sur l'état syntaxique de l'élément qui la suit et son emploi devrait être rendu obligatoire. Cette virgule peut toujours être remplacée par le mot ET. Mais cette conjonction ayant un rôle logique dans les conditions composées, son emploi comme connectif facultatif est peu heureux.

Les prépositions telles DE, A, SUR, POUR, sont, selon les clichés, facultatives ou obligatoires et dans ce dernier cas auront des significations différentes suivant le verbe auquel elles se rapportent. Par exemple DE correspond à un signe - dans SOUSTRAIRE...DE.., à un signe = dans CALCULER...DE.., à une introduction de données dans RECEVOIR...DE... On aura donc au moment de la compilation à les différencier soit en leur donnant des codes différents, soit en conservant l'état syntaxique qu'elles représentent en mémorisant le verbe pendant la durée de sa portée.

Pour l'instant nous nous sommes peu rendu compte de la longueur d'écriture de la division TRAITEMENT. Pourtant dans les exemples précédents nous avons vu que pour réaliser l'opération

$$W = \frac{(X + Y) \times Z}{T}$$

il nous avait fallu écrire trois verbes arithmétiques. En fait l'instruction

CALCULER W = (X + Y) * Z / T

aurait suffi mais est assez contraire à l'esprit COBOL le verbe CALCULER n'étant d'ailleurs pas dans les spécifications du COBOL de base. Si un compilateur ne possède pas ce verbe, le simple calcul de X^3 devient un véritable petit problème car il faut non seulement deux opérations MULTIPLIER, mais encore se définir dans la section INTERMEDIAIRES, une mémoire isolée pouvant stocker le résultat intermédiaire. X^2, au cas où X^3 devrait être édité.

Les autres verbes appellent peu de commentaires si ce n'est le verbe TRANSFERER dont les règles devront être définies pour chaque machine en accord avec les clauses d'édition, de CLASSE et d'USAGE.

Certaines abréviations sont autorisées pour rendre la programmation moins lourde ceci surtout dans les conditions composées. Aussi l'expression booléenne

(A > B) ET (A > C) OU A = D

pourra s'écrire :

A > B ET < C OU = D

ceci étant notoirement utile si A est un identificateur de 8 lettres nécessitant deux qualifications.

Par contre avec ces abréviations on peut arriver à des phrases correctes mais devenues incompréhensibles. Par exemple :

A = B ET (C OU D ET E > Y) OU < Z

qui signifie en réalité

(A = B ET (A = C OU (A = D ET E > Y))) OU E < Z

Or à première vue, il n'était pas évident que le dernier signe < portait sur E et non sur A.

4. 3. Rapidwrite. [9]

Pour pallier aux inconvénients de l'écriture lourde de COBOL, un constructeur anglais (I. C. T.) a préconisé une écriture rapide et condensée de COBOL, nommée Rapidwrite. Ce langage ne traiterait pas la division IDENTIFICATION et dans les autres divisions ne traiterait que :

. Pour l'EQUIPEMENT : les capacités de mémoire et les implantations sur milieu externe.

. Pour les DONNEES : les options MODELE, REDEFINIT, FIGURE, VALEUR.

. Pour le TRAITEMENT : les étiquettes, les indices, les verbes et conjonctions CALCULER, SI, ALLER, EXECUTER, TRANSFERER, LIRE, ECRIRE, INCLURE, ARRET.

Des cartes spéciales seraient imprimées pour chacune des instructions ou déclarations.

On obtient ainsi un langage orienté vers les applications commerciales, synthétique et relativement proche du COBOL. L'inconvénient majeur est que la compréhension du programme pour un non-initié sera pratiquement impossible. De plus des cartes de formats spéciaux devraient être employées et il n'est plus question de compatibilité. Par contre il semble possible d'inclure avant la phase de réduction pour la compilation (codification) un dictionnaire de synonymes pour les noms de données et de traitement et un dictionnaire de formats spéciaux tels que le programme écrit en COBOL puisse être édité avant même les diagnostics relatifs à la compilation.

5. COMPARAISON AVEC DES AUTOCODES COMMERCIAUX. |9|.

La compartimentation très stricte en divisions et sections à buts explicitement définis n'est pas propre à COBOL.

Ceci est commun à tous les autocodes commerciaux, mais jamais à un point aussi grand que dans COBOL.

La division IDENTIFICATION apparaît toujours d'une manière ou d'une autre, sous forme de titre du programme. Nous n'y reviendrons pas.

5. 1. Division EQUIPEMENT.

C'est dans COBOL qu'apparaît pour la première fois une division spécifique à la description du matériel utilisé, lors du programme (compilation plus exécution). Dans les autres autocodes commerciaux, cette description est noyée dans le reste du programme. Ceci est parfaitement compréhensible puisqu'aucun de ces langages n'avait de prétention à des communications inter-machine et qu'ils n'étaient écrits qu'en fonction d'un type d'ordinateur particulier et déterminé (FACT et NEBULA) ou d'une série de machines construites suivant les mêmes normes (COMTRAN, FLOWMATIC).

Nous avons vu ce qu'il y avait d'intéressant au point de vue analyse du problème, dans cette division EQUIPEMENT qui est, à n'en pas douter l'une des parties les plus solides de COBOL et dont d'autres langages, même scientifiques tels qu'ALGOL pourraient s'inspirer.

5. 2. Division DONNEES.

Dans COMTRAN, l'indication de support externe de fichier se fera dans la division DONNEES par la seule indication F, C, W correspondant aux trois sections de la division DONNEES. On suppose donc que tous les fichiers qu'ils soient sur cartes ou sur bandes auront des structures identiques. Dans NEBULA, cette indication se fait également dans la division DONNEES et en FACT dans la division TRAITEMENT. Il y a donc rationnalisation dans l'écriture du programme COBOL, mais non condensation dans la compilation car au moment de la rencontre de l'un des verbes se rattachant à une opération d'entrée-sortie, il faudra savoir de toute façon sur (ou depuis) quel organe périphérique on doit travailler.

La première remarque que l'on puisse faire sur la divi-

sion DONNEES de COBOL, c'est que contrairement aux divisions équivalentes des autres autocodes, elle ne nécessite aucune feuille de programmation spéciale. Le format de référence est très peu restrictif et une liberté quasi-totale est laissée au programmeur pour la représentation de son programme. D'autre part l'ordre des clauses d'indications d'entrée est indifférent, ce qui du point de vue programmation laisse un excellent degré de liberté, mais ce qui du point de vue compilation nécessitera des efforts accrus.

COBOL distingue d'une façon nette la description physique et la description logique des fichiers. On peut cependant lui reprocher l'emphase mise sur les fichiers écrits sur bandes, en effet tout ce qui est relatif aux repères n'est défini qu'en fonction d'eux. Mais cela est réalisé implicitement car, en fait, il n'y a pas de modèle descriptif spécialisé pour chaque support externe. Celui-ci ayant été défini dans la division EQUIPEMENT il ne reste plus qu'à donner des renseignements sur la géographie du fichier. Ceci est beaucoup moins lourd que dans FLOWMATIC ou NEBULA qui avec leurs modèles à structure fixe, empêchent toute compatibilité, ou plus complet que dans COMTRAN qui ne décrit rien puisque les fichiers IBM ont tous une structure identique. Quant à FACT rien n'apparaît pour le non-initié et il semble que les seules descriptions complètes sont celles de Fichiers sur cartes.

La structure des fichiers commerciaux impose une description logique s'appuyant sur le concept de niveaux. Cette structure d'arbre, beaucoup trop étendue dans COBOL avec ses 49 niveaux est par contre trop restrictive dans FLOWMATIC et NEBULA avec quatre niveaux. Dans FACT ces niveaux n'existent que dans disposition de mise en page du programmeur, ce qui empêche les redéfinitions ou chevauchement de zones qui n'existent d'ailleurs que dans COBOL et COMTRAN. Ce dernier autorise 9 niveaux seulement du fichier au mot élémentaire et cela est amplement suffisant. Le seul inconvénient de la descript on COMTRAN est que les niveaux sont déjà préimprimés sur la feu lle de programmation et ne signifient pas grand chose au lecteur du programme. Le mieux aurait été de garder la mise en page COBOL, avec 9 niveaux possibles, ces niveaux étant

écrits plutôt en chiffres qu'en lettres afin d'avoir une signification évidente.

Dans les autres autocodes seuls les mots élémentaires sont décrits, généralement sous forme de modèle, et de ce fait tous les cadrages spéciaux ne sont pas réalisables. Cependant, mis à part COMTRAN, le nombre d'options est inférieur à l'étendue que présente COBOL.

COBOL autorise trois indices ce qui est tout à fait suffisant pour un problème commercial. Dans FACT et COMTRAN un indice est autorisé et aucun dans FLOWMATIC et NEBULA. Les indices COBOL doivent être des nombres entiers ou des noms de données à valeurs entières alors que FACT autorise des expressions arithmétiques. La qualification par un suffixe existe dans COMTRAN et FLOMMATIC, mais limitée au fichier elle interdit de donner deux noms identiques à des mots du même fichier.

5. 3. Division TRAITEMENT.

Tous les autocodes commerciaux ont les verbes AJOUTER et SOUSTRAIRE. L'option de COBOL, RESULTAT est telle qu'elle évite d'employer le verbe CALCULER ou un verbe de transfert comme dans FLOWMATIC, COMTRAN, FACT et NEBULA. Les verbes MULTIPLIER et DIVISER n'existent pas en COMTRAN mais, par contre ce dernier comprend la fonction valeur absolue qui aurait dû être un verbe COBOL. Ainsi si l'on veut, en COBOL prendre la valeur absolue d'un nombre, il faut écrire :

SI X POSITIF ALORS TRANSFERER X EN X SINON CALCULER Y = - X.

Les ARRONDIS et tronçonnements sont possibles ou automatiques dans tous ces autocodes sauf FLOWMATIC.

Les opérations d'entrée-sortie sont à peu près semblables pour tous les autocodes commerciaux. Seules peuvent différer les opérations de servitude réservées à chaque verbe, c'est-à-dire verrouillage, rebobinage etc... qui sont décrites plus complètement dans COBOL. Cependant il manque à COBOL deux fonctions ma-

jeures très employées en gestion commerciale : l'édition des états et le tri. Dans FACT, il est possible de préparer directement une ligne d'état à imprimer, avec tous les masques et insertions de caractères utiles. En COBOL, on doit programmer ces insertions par des MODELES de la division DONNEES, et faire des transferts successifs. Cette lacune sera prochainement comblée puisqu'une section ETATS va être ajoutée à la division DONNEES de COBOL 61. Pour le tri, il n'est pas prévu de verbe ou fonction spéciale. Cette omission pourra être réparée puisqu'une option TRIE SUR existe déjà dans la description de fichier.

COBOL présente la gamme de verbes de branchement la plus complète. Le branchement ALLER A peut-être indicié comme dans COMTRAN. L'exécution de boucles est possible avec trois niveaux d'indices comme dans COMTRAN alors qu'un seul indice est utilisé en FACT et aucun dans NEBULA. Dans FLOWMATIC les boucles doivent être programmées.

Dans COBOL, il n'existe qu'un verbe pour toutes les opérations de transfert que ce soit groupe-groupe ou élémentaire-élémentaire. Puisqu'il n'existe pas encore de section "ETATS", toutes les opérations d'édition se feront avec ce verbe. Dans FACT et NEBULA, il existe un verbe de remise à zéro tandis que dans COBOL, il faudra écrire : TRANSFERER ZERO EN ... Dans FACT également, il est possible de remplacer un fichier complet par un autre en exceptant certains articles. Par contre l'opération de remplissage TRANSFERER TOUS... n'existe qu'en COBOL et FLOWMATIC. Le verbe EXAMINER n'existe également que dans COBOL mais on voit mal son intérêt.

Il est possible de définir des fonctions dans la division TRAITEMENT de COBOL ; ces fonctions seront de nouveaux verbes avec formats à paramètres formels. Dans COMTRAN, on définit de nouveaux noms résultats d'une procèdure à paramètres formels. Ceci est possible aussi dans la division DONNEES de NEBULA, mais rien de ce genre n'apparait ni dans FACT, ni dans FLOWMATIC.

En conclusion, COBOL présente des caractères plus complets

et des facilités de programmation plus étendues que les autocodes dont il est le successeur logique (COMTRAN et FLOWMATIC). La comparaison avec NEBULA lui est favorable du même point de vue. Il est plus difficile d'en dire autant avec FACT, mais la difficulté de lecture laisse un préjugé défavorable à celui qui non-initié, voudrait utiliser FACT rapidement. D'autre part la recherche de compatibilité n'existant qu'en COBOL, fournit un nouvel argume t de supériorité pour ce dernier.

6. AVANTAGES ET INCONVENIENTS DE COBOL, LANGAGE COMMERCIAL.

On peut, en conclusion de ce chapitre, citer les avantages suivants pour COBOL :

a) Compréhension du langage et d'un programme écrit en COBOL sans avoir une notion du langage machine.

b) Diminution du coût de la programmation, celle-ci étant plus facile et les mises au point au niveau du langage source plus rapides.

c) Certaines erreurs d'inattention seront exclues par le fait même d'un langage compréhensible et peu formel.

d) Les programmes pourront être modifiés facilement et des additions pourront être incluses sans difficulté de programmation. Le repérage se faisant au niveau usuel ; il n'y aura pas à rechercher des codes machine, lors de modifications éventuelles.

e) Le domaine d'utilisation que recouvre COBOL est suffisament vaste pour permettre de résoudre la majeure partie des problèmes de gestion.

f) Les programmeurs seront entrainés plus facilement, la connaissance du langage étant rapidement assimilable.

g) La compatibilité, même partielle, entre différentes machines

h) Moyen d'analyse rationnel d'un problème.

i) Progrès sur les langages de programmation existant.

Par contre, on rencontre les difficultés et inconvénients suivants :

a) La difficulté de la synthétisation de l'analyse subsiste.

b) La connaissance au moins partielle de la machine est né-

cessaire pour avoir un programme objet efficace (segmentation, connaissance des programmes standards, forme interne et externe des données etc...)

c) Une définition stricte des données et formats exacts est nécessaire. Un grand nombre de clichés comporte des mots superficiels qui devraient être supprimés.

d) Le manque de formalisme du langage, surtout dans la division DONNEES, amenant une compilation très lourde.

e) Une définition des données trop orientée vers les machines à caractères ce qui entraine une perte de compatibilité dans cette division.

f) Enfin et surtout, l'efficacité du programme généré. Nous verrons dans le chapitre 4 ce que l'on peut penser, mais en tous cas le temps de compilation est grand et l'exécution ne pourra jamais être aussi optimisée que dans un programme écrit directement.

o°o

III. COMPARAISON D'ALGOL ET DE COBOL.

1. INTRODUCTION.

Depuis qu'ALGOL |10|, |11|, |12|, et COBOL sont devenus les deux langages les plus aptes à la standardisation, l'un pour le calcul scientifique, l'autre pour la gestion commerciale, de nombreux essais ont été tentés pour les fondre en un seul langage |13|. Avant de les comparer, il est intéressant d'essayer de formaliser COBOL, c'est-à-dire de décrire par exemple sa syntaxe suivant la métalangue utilisée pour ALGOL. Ceci a déjà été réalisé pour les concepts très généraux de la division TRAITEMENT par J. SAMMET |14|, et des définitions de variables métalinguistiques apparaissent également implicitement dans une carte syntaxique de COBOL |15|, décrite suivant la métalangue utilisée pour le rapport COBOL.

2. COMPARAISON DES METALANGUES ALGOL ET COBOL.

2. 1. Métalangue ALGOL.

La métalangue ALGOL est composée de trois symboles de base et de deux concepts.

Les symboles sont :

::= signifiant "défini par"

| signifiant "ou" d'énumération

< > crochets de délimitation qui bornent une suite de caractères représentant les variables métalinguistiques.

Les concepts sont :

--- variables métalinguistiques : suite de caractères entre " < " dont les valeurs sont des suites de symboles.

-- opération de concaténation.

Ainsi <ab> ::= (|)| <ab>(|<ab><d>

définit la variable métalinguistique ab .

En supposant que les valeurs de d sont les chiffres décimaux, des valeurs possibles pour ab sont :

((((46(3
))( )246(3

alors que () ou 3(ne sont pas possibles.

2. 2. Métalangue COBOL.

La métalangue COBOL est composée de quatre symboles de base et trois caractéristiques d'édition.

Les symboles sont :

{ } signifiant qu'un choix doit être fait parmi le contenu.

[ ] " " " peut " " " " "

soulignant les mots clés du langage.

... signifiant que le dernier type de chose écrite peut-être répétée indéfiniment.

Les caractéristiques d'édition sont :

-- mots en minuscules sont des termes génériques dont les valeurs spécifiques doivent être données par l'utilisateur

-- mots en majuscules soulignés sont les mots clés du langage.

-- mots en majuscules non soulignés sont des mots facultatifs du langage.

Ainsi :

$$\underline{\text{SOUSTRAIRE}} \left\{\begin{matrix}\text{libellé-1}\\ \text{donnée-1}\end{matrix}\right\} \left[\left[\begin{matrix},\\ \text{ET}\end{matrix}\right] \left\{\begin{matrix}\text{libellé-2}\\ \text{donnée-2}\end{matrix}\right\} \ldots\right] \underline{\text{DE}} \left\{\begin{matrix}\text{libellé-n}\\ \text{donnée-n}\end{matrix}\right\} \left[\underline{\text{RESULTAT}}\ \text{donnée-n}\right]$$

pourra donner entre autres possibilités :

SOUSTRAIRE donnée-1, donnée-2, libellé-1 DE libellé-2 RESULTAT donnée-3

SOUSTRAIRE libellé-1 DE donnée-1

Mais non :

SOUSTRAIRE donnée-1, donnée-2 RESULTAT donnée-3

SOUSTRAIRE donnée-1 ET DE donnée-2

2. 3. Comparaison.

Il est évident que | pour ALGOL et { } pour COBOL ont la

même signification de connectifs de choix obligatoire. D'autre part les mots facultatifs de COBOL peuvent facilement être exprimés comme une série de connectifs. Dans les formules métalinguistiques de COBOL AB et A[B] sont équivalentes et peuvent encore être écri- $\left.\begin{matrix}A\ B\\ A\end{matrix}\right\}$ ou suivant la métalangue ALGOL A | AB .

A noter que A [B] a encore la même signification, mais B est en fait une redondance de définition de mot facultatif (facultatif d'une part parce que non souligné, d'autre part mis entre crochets). En fait on employera généralement AB lorsque B ne peut apporter aucune action et A [B] lorsque B implique un supplément à l'action A.

La différence majeure vient de ce qu'en COBOL on ne peut assigner de nom à une variable métalinguistique particulière et que par conséquent, les variables ne peuvent être définies récursivement, bien que cela soit fait implicitement dans les cas simples par le symbole ...

Ainsi $\left\{\begin{matrix}X\\ Y\end{matrix}\right\}[Y\ \ldots]$ et $\langle a\rangle ::= X \mid Y \mid \langle a\rangle\ Y$ sont équivalents.

Dans ce qui précède nous n'avons pas tenu compte des espaces qui ont une signification typographique en COBOL et qui de ce fait rendent les opérations de concaténation très longues à écrire si l'on veut suivre rigoureusement les règles d'écriture COBOL. Dans ce qui suit nous ne les représenterons pas puisqu'ils ne jouent aucun rôle syntaxique autre que délimiteur de mots. De même les connecteurs tels "," "ET" ";" " LORS" seront indiqués tels qu'ils apparaissent après transformation du langage source en chaine codée (cf II. 4. 2.), et les abréviations autorisées dans les conditions composées ne seront pas traitées.

2. 4. Exemple.

Nous pouvons pour illustrer ce que nous venons de voir, reprendre les exemples donnés et les écrire suivant la métalangue de l'autre langage.

Ainsi $\langle ab\rangle$ pourra s'écrire en COBOL $\left\{\begin{matrix}(\\ )\end{matrix}\right\}[[(][)]\ldots]$

en notant que d représente également une variable métalinguistique.

Pour l'exemple COBOL, il faut définir des variables métalinguistiques intermédiaires :

<opérande> ::= <donnée> | <libellé>
<séparateur de liste> ::= , | ET
<liste d'opérande> ::= <opérande> | <liste d'opérande> <séparateur de liste> <opérande>
<SOUSTRAIRE gauche> ::= SOUSTRAIRE <liste d'opérande> DE
<option résultat> ::= <donnée> | <opérande> RESULTAT <donnée>
<opération SOUSTRAIRE> ::= <SOUSTRAIRE gauche> <option résultat>

Il faut remarquer l'intérêt de cette notation. En effet dans la notation métalinguistique de COBOL pour cette opération SOUSTRAIRE, une note sémantique doit être ajoutée; indiquant que dans le cas où le mot RESULTAT n'est pas employé, le terme suivant DE ne peut être un libellé. Dans la notation métalinguistique d'ALGOL cette note n'a plus de raison puisqu'elle est implicitement exprimée par la variable métalinguisitque "option résultat".

3. DEFINITION DE LA DIVISION TRAITEMENT DE COBOL SUIVANT LA METALANGUE ALGOL.

En fait, de par la définition de symboles de base, types de mot etc... nous débordons nécessairement de la division TRAITEMENT et donnons une définition des concepts généraux.

3. 1. Symboles de base.

<symbole de base> ::= <lettre> | <chiffre> | <délimiteur> | <éditeur>
<lettre> ::= A | B | ... | Z
<chiffre> ::= 0 | 1 | ... | 9
<éditeur> ::= , | . | * | $
<délimiteur> ::= <opérateur> | <séparateur>
<opérateur> ::= <opérateur arithmétique> | <opérateur de relation>

<opérateur arithmétique> ::= + | - | * | /
<opérateur de relation> ::= = | < | >
<séparateur> ::= <séparateur d'instruction> | <séparateur de mot>
<séparateur d'instruction> ::= . | ; | □
<séparateur de mots> ::= <séparateur d'instruction> | (| , |) | "

Remarque : □ indique un espace.

3. 2. Types de mots.

<mot> ::= <lettre> | <chiffre> | <mot> <lettre> | <mot> - <mot> | <mot> <chiffre>
<type de mot> ::= <nom> | <verbe> | <mot réservé>
<nom> ::= <nom de donnée> | <nom de traitement> | <libellé> | <nom de condition> |
<nom de registre spécial> | <noms spéciaux>
<verbe> ::= Un des 23 verbes de COBOL
<mot réservé> ::= <mot clé> | <mot facultatif>
<mot clé> ::= tout mot en majuscules soulignés apparaissent dans un cliché
<mot facultatif> ::= tout mot en majuscules non soulignés apparaissent dans un cliché
<mot terminé de base> ::= <mot> <séparateurs de mots>
<mot terminé> ::= <mot terminé de base> □ | <mot terminé> □

Remarque : Dans la suite nous ne tiendrons plus compte des séparateurs de mots lorsqu'il s'agit de blancs et nous supposerons tous les mots comme terminés.

Un mot ne peut avoir plus de 30 caractères (impossible pratiquement à décrire suivant cette métalangue).

3. 3. Les noms :

<identificateur> ::= <lettre> | <identificateur> <identificateur> | <id> <chiffre> | <chiffre> <id> | <id> - <id> | <chiffre> - <id> | <id> - <chiffre>
<entier sans signe> ::= <chiffre> | <entier sans signe> <chiffre>

Remarque : on utilise id comme abréviation d'identi-

ficateur.

3.3.1. Nom de donnée (Abrév. ndd)

<ndd> ::= <variable simple> | <variable indiciée>
<nom de fichier> ::= <id>
<nom d'article> ::= <id> | <id> DANS <nom de fichier>
<nom de mot> ::= <id> | <id> DANS <variable simple>
<variable simple> ::= <nom de fichier> | <nom d'article> | <nom de mots>. Remarque : Le nombre de récursivités pouvant porter sur DANS est limité à 50 (nombre de niveaux possibles)

<indice> ::= <nombre entier>
<nombre entier> ::= <nom de donnée à valeur entière> | <entier sans signe>
<liste d'indice> ::= <indice> | <indice>, <indice> | <indice>, <indice>, <indice>
<variable indiciée> ::= <variable simple> (<liste d'indices>

3. 3. 2. Nom de traitement.

<étiquette> ::= <id> | <entier sans signe>
<nom de paragraphe> ::= <étiquette>
<nom de section> ::= <étiquette>
<nom de traitement> ::= <nom de section> | <nom de paragraphe> DANS <nom de section> | <nom de paragraphe>

3. 3. 3. Nom de condition.

<nom de condition> ::= <id> | <id> DANS <nom de mot>

3. 3. 4. Libellés (Abbrev. lib)

<lib> ::= <constante figurative> | <lib alphanumérique> | <lib numérique>
<constante figurative> ::= liste donnée dans le rapport COBOL | TOUS " <lib> "
<lib alphanumérique> ::= " <chaine symbole> "
<chaine symbole> ::= <symbole de base> | <chaine symbole> <symbole de base>
<lib numérique> ::= + <nombre décimal> | - <nombre décimal> | <nombre décimal>
<nombre décimal> ::= <entier sans signe> | <fraction déci-

male > | <entier sans signe > <fract dec >
<fraction décimale >::= . <entier sans signe >

Remarque : Dans la " chaine symbole " le symbole " ne peut apparaître.

3. 3. 5. Nom de registre spécial.

<nom de registre spécial >::= COMPTEUR

3. 3. 6. Noms spéciaux.

<noms spéciaux > ::= <nom mnénonique>
<nom mnénonique > ::= <id>

3. 4. Formules.

<opérateur d'addition> ::= + | PLUS | - | MOINS
<opérateur de multiplication >::= * | MULTIPLIE PAR | FOIS | / | DIVISE PAR
<opérateur de puissance > ::= PUISSANCE | * *
<primaire arithmétique > ::= <n d> | <nombre décimal> | (< expression arithmétique simple >)
<facteur > ::= <primaire arithmétique > | < facteur > <opérateur de puissance > <primaire arithmétique>
<terme > ::= <facteur > | < terme > < opérateur de multiplication > < facteur >
<expression arithmétique simple > ::= < terme > | < opérateur d'addition > <terme > | < expression arithmétique simple > <opérateur d'addition > < terme>
<formule > ::= <expression arithmétique simple >

3. 5. Expressions booléennes.

3. 5. 1. Relation.

<objet >::= <formule > | < lib >
<sujet > ::= <objet >
<opérateurs de relation complets > ::= <opérateurs de relation> |
<op. de relation alphabétique>
<opérateurs de relation alphabet > ::= DIFFERENT | DEPASSE |
<Opérateurs d'égalité alphabet > | NON <opérateurs d'égalité alphabet>
<opérateurs d'égalité alphabet > ::= SUPERIEUR | INFERIEUR | EGAL

<relation> ::= <sujet> <opérateurs de relation complets> <objet>

3. 5. 2. Test de zéro.

<comparateur de zéro> ::= POSITIF | NEGATIF | ZERO

<opérateur de zéro> ::= NON <comparateur de zéro> | <comparateur de zéro>

<test de zéro> ::= <formule> <opérateur de zéro>

3. 5. 3. Test de classe.

<comparateur de classe> ::= ALPHABETIQUE | NUMERIQUE

<opérateur de classe> ::= <comparateur de classe> | NON <comparateur de classe>

<test de classe> ::= <ndd> <opérateur de classe>

3. 5. 4. Terme booléen.

<primaire booléen> ::= <nom de condition> | <noms spéciaux> | <test de classe> | <test de zéro> | <relation> | (<terme booléen>)

<secondaire booléen> ::= <primaire booléen> | NON <primaire booléen>

<facteur booléen> ::= <secondaire booléen> | <facteur booléen> ET <secondaire booléen>

<terme booléen> ::= <facteur booléen> | <terme booléen> OU <facteur booléen>

Remarque : NON précède une parenthèse droite ou un primaire booléen ne comportant pas de NON.

3. 5. 5. Expression booléenne.

<expression booléenne> ::= <terme booléen>

3. 6. Instructions impératives.

(Les mots facultatifs ne sont pas indiqués).

3. 6. 1. Instruction finale.

<inst. ALLER> ::= ALLER <nom de traitement> | ALLER <liste de noms de traitement> SELON <ndd>

<liste de noms de traitement> ::= <nom de traitement> | <liste de nom de tr.> , <nom de tr.>

<instruction finale> ::= <instr ALLER> | PHRASE SUIVANTE | ARRET FINAL

3. 6. 2. Instruction-E/S.

<liste de noms de fichier> ::= <nom de fichier> | <liste de noms de fichier>, <nom de fichier>
<clause SORTIE> ::= SORTIE <liste de noms de fichier>
<liste de noms de fichier entrée> ::= <liste de noms de fichier> INVERSE | <liste de noms de fichier entrée>, <liste de noms de fic entrée> | <liste de noms de fichier>
<clause entrée> ::= ENTREE <liste de noms de fichier entrée>
<instr OUVRIR> ::= OUVRIR <clause entrée> | OUVRIR <clause sortie> | OUVRIR <clause entrée> <clause sortie>
<option verrou> ::= VERROUILLER | SANS REBOBINER
<clause fermeture> ::= <nom de fichier> <nom de fichier> <option verrou> |
<nom de fichier> BOBINE | <nom de fichier> BOBINE <option verrou>
<liste de fermeture> ::= <clause de fermeture> | <liste de fermeture> <clause de fermeture>
<inst FERMER> ::= FERMER <liste de fermeture>
<clause ff> ::= FIN-DE-FICHIER <instruction impérative> | FIN-DE-FICHIER <instruction impéra> SINON <instruction>
<instr LIRE simple> ::= LIRE <nom de fichier>
<instr LIRE transfert> ::= <instr LIRE simple> TRANSFERT <nom d'article>
<instr LIRE incond> ::= <instr LIRE simple> | <instr LIRE transfert>
<instr LIRE> ::= <instr LIRE incond> | <instr LIRE incond>
<séparateur d'instruction> <clause ff>
<interlignes> ::= <nom mnémonique> | <entier sans signe>
<option saut> ::= AVANT <interlignes> | APRES <interlignes>
<instr ECRIRE simple> ::= ECRIRE <nom d'article>
<instr ECRIRE transfert> ::= <instr ECRIRE simple> DEPUIS <ndd>

<instr ECRIRE normale> ::= <instr ECRIRE simple> | <inst ECRIRE transfert>

<instr ECRIRE> ::= <instr ECRIRE normale> <option saut> | <instr ECRIRE normale>

<nom d'indication> ::= <ndd> | <lib>

<liste de noms d'indic> ::= <nom d'indic> | <liste de noms d'indic> , <nom d'indic>

<instr INDIQUER> ::= INDIQUER <liste de noms d'indication> | INDIQUER <liste de noms d'indic> SUR <nom mnémonique>

<instr RECEVOIR> ::= RECEVOIR <ndd> | RECEVOIR <ndd> DE <nom mnémomique>

<instruction-E/S> ::= <instr OUVRIR> | <instr FERMER> | <instr LIRE> | <instr ECRIRE> | <instr INDIQUER> | <instr RECEVOIR>

3. 6. 3. Instruction arithmétique.

<opérande> ::= <ndd> | <lib numérique>

<liste opérande> ::= <opérande> | <liste opérande> , <opérande>

<option résultat> ::= <ndd> | <opérande> RESULTAT <ndd>

<option DC> ::= DC <instruction impérative> | DC <instruction impérative> SINON <instruction>

<partie droite arithmétique> ::= ARRONDI | <séparateur d'instruction> <option DC> | ARRONDI <séparateur d'instruction> <option DC>

<addition> ::= AJOUTER <liste opérande> <option résultat>

<instr AJOUTER> ::= <addition> | <addition> <partie droite arith>

<soustraction> ::= SOUSTRAIRE <liste opérande> DE <option résultat>

<instr SOUSTRAIRE> ::= <soustraction> | <soustraction> <partie droite arith>

<multiplication> ::= MULTIPLIER <opérande> PAR <option résultat>

<instr MULTIPLIER> ::= <multiplication> | <multiplication> <partie droite arith>.

<division> ::= DIVISER-PAR <opérande> QUANTITE <option ré-

sultat >
<instr DIVISER> ::= <division>| <division> < partie droite arith>
<opération d'assignation> ::= = | ARRONDI =
<calcul> ::= CALCULER <ndd> <opération d'assignation> <formule>
<instr CALCULER> ::= <calcul> | <calcul> < option DC>
<instr arithmétique> ::= <instr AJOUTER> | < instr SOUSTRAIRE> | <instr MULTIPLIER> | <instr DIVISER> | <instr CALCULER>

3. 6. 4. Instruction de branchement.

<branchement> ::= <nom de traitement> POUR PASSER A <nom de traitement>
<liste de branchement> ::= <branchement> | <liste de branchement>, <branchement>
<instr CHANGER> ::= CHANGER <liste de branchement>
<partie gauche EXECUTER> ::= EXECUTER <nom de traitement> | EXECUTER nom de traitement> JUSQUA <ndd>
<clause execution> ::= LIMITE <expression booléenne> | < entier sans signe> FOIS | <liste exécution>
<liste exec primaire> ::= AVEC <ndd> DE <opérande> PAR <opérande> LIMITE <express booléenne>
<liste exec secondaire> ::= APRES <ndd> DE <opérande> PAR <opérande> LIMITE <express booléenne>
<liste exécution> ::= <liste exec primaire> | < liste exec primaire> <liste exec secondaire> | <liste exec primaire> <liste exec secondaire> < liste exec secondaire>
<instr EXECUTER> ::= <partie gauche EXECUTER> | < partie gauche EXECUTER> <clause execution>
<instruction de branchement> ::= <instr CHANGER> | <instr EXECUTER>

3. 6. 5. Instruction de mouvement.

<donnée> ::= <ndd> | <lib>
<liste de ndd> ::= <ndd> | <liste de ndd>, <ndd>
<instr TRANSFERER simple> ::= TRANSFERER <donnée> EN <liste de ndd>

<instr TRANSFERER concordance> ::= TRANSFERER CONCORDANCE < ndd> EN <liste ndd>

<instr TRANSFERER> ::= <instr TRANSFERER simple> | <instr TRANSFERER concordance>

<option compteur> ::= ARRETE PREMIER | TOUS | PREMIERS

<option remplaçant> ::= <option compteur> | PREMIER

<clause compteur> ::= COMPTER < option compteur> <libellé> | COMPTER <option compteur> < lib> REMPLACER PAR < lib>

<clause remplaçant> ::= REMPLACER <option remplaçant> PAR <lib>

<partie droite EXAMINER> ::= <clause compteur> | <clause remplaçant>

<instr EXAMINER> ::= EXAMINER <ndd> <partie droite EXAMINER>

<instr mouvement> ::= <instr TRANSFERER> | <instr EXAMINER>

3. 6. 6. Instruction arret.

<instr ARRET> ::= ARRET < lib>

3.6. 7. Instruction opérationnelle.

<instr opérationnelle> ::= <instr-E/S> | < instr arithmétique> | < instr de branchement> | <instr de mouvement> | <instr d'arrêt>

3. 6. 8. Instruction inconditionnelle.

<instr incond> ::= < instr opérationnelle> | < instr incond> <séparateur d'instruction> < instr opérationnelle>

3. 6. 9. Instruction impérative.

<instr impérative> ::= <instr incond> | < instr incond> <séparateur d'instr> <instr finale> | <instr finale>

3. 7. Instruction conditionnelle.

<clause SI> ::= SI <expression booléenne>

<instr SI> ::= < clause SI> ALORS <instruction>

< instr conditionnelle> ::= <instruction SI> | < instruction SI> SINON <instruction> .

3. 8. Section.

<instruction> ::= < instr impérative > | < instr conditionnelle> |
<instr incond > < séparateur instr > < instr conditionnelle >
<phrase > ::= <instruction> .
<paragraphe non étiquetté > ::= <phrase> | <paragr. non étiquetté > <phrase>
paragraphe étiquetté ::= <étiquette > < paragraphe non étiqueté>| < étiquette> CONTINUER. |< étiquette > NOTER <chaine de symbole> . |< étiquette> ENTRER <langage>
< paragraphe > ::= <paragraphe non étiqueté > | <paragraphe étiqueté >
<en-tête de section> ::= <étiquette> SECTION. | <étiq> <nbre priorité> SECTION
section ::= en-tête de section paragraphe section paragraphe
<nbre priorité> ::= <chiffre> | <chiffre> <chiffre>

Remarque : Dans la chaine de symbole suivant NOTER, un "." ne peut apparaître.

La variable "langage" suivant ENTRER indique des procédures en code.

3. 9. Section DECLARATIONS.

3. 9. 1. Section INCLURE.

<paramètre formel> ::= < id >
<routine > ::= < paramètre formel > PAR < donnée>
<liste de routine > ::= <routine>| < liste de routine >, < routine>
<instr INCLURE > ::= INCLURE < nom de traitement>| INCLURE <nom de traitement > < clause routine >
<clause routine > ::= REMPLACER <liste de routine>
<section INCLURE > ::= <en tête de section > < instr INCLURE >

3. 9. 2. Section DEFINIR.

<liste de paramètres formels > ::= < paramètre formel>| <liste de de paramètres formels>, <par. formel >
<nouveau-verbe > ::= < id >
<instr DEFINIR> ::= DEFINIR <nouveau-verbe> FORME <nouveau-verbe > (<liste de paramètres formels >) . <paragraphe >
<section DEFINIR > ::= < enttête de section > < instr DEFINIR>

3. 9. 3. Section UTILISER.

<option sélection > ::= ENTREE| SORTIE |<nom de fichier>

<option temps> := APRES | AVANT
<option moment> ::= DEBUT | FIN
<option support> ::= BOBINE | FICHIER
<option repère> ::= <option moment> | <option support> |
<option moment> <option support>
<instr UTILISER gauche> ::= UTILISER APRES ERREUR | UTILISER
<option temps> STANDARD | UTILISER <option temps> <option repère> STANDARD
<instr UTILISER> ::= <instr UTILISER gauche> <option sélection>
<section UTILISER> ::= <en-tête de section> <instr UTILISER> <paragraphe>

3. 9. 4. Section DECLARATIONS.

<section de procèdure> ::= <section INCLURE> | <section DEFINIR> | <section UTILISER>

<déclaration de procédure> ::= <section de procédure> |
<déclaration de proc> <section de proc>
<section DECLARATIONS> ::= DECLARATIONS <déclaration de procedure> BORNE DECLARATIONS.

3. 10. Division TRAITEMENTT.

<division TRAITEMENT> ::= TRAITEMENT. <section DECLARATIONS> <section> | TRAITEMENT. <section>

4. EXTENSION POSSIBLE AUX AUTRES DIVISIONS DE COBOL.

La description des divisions IDENTIFICATION et EQUIPEMENT suivant la métalangue ALGOL pourrait-être faite assez facilement. Mais les possibilités de récursion étant faibles et le nombre d'options possibles grand, cela présente peu d'intérêt et la métalangue COBOL est plus claire.

Dans la division DONNEES, les remarques sémantiques sont extrèmement importantes lorsqu'il s'agit de décrire la struc-

ture d'arbre, les notions d'article, de mots groupes et élémentaires, les noms de condition etc;.. D'autre part à moins de faire des pages d'écriture, il n'est pas possible de décrire d'une manière simple les clauses possibles et non contradictoires servant à décrire tel ou tel type de mot. On serait amené à définir des variables métalinguistiques telles que " mot élémentaire numérique de longueur fixe " , " mot élémentaire alphanumérique sans clauses d'édition " etc... Le seul intérêt serait alors d'introduire ces variables dans les définitions de variable de la division TRAITEMENT. Mais à ce moment-là, on se heurterait à un autre danger, celui de l'incompatibilité. Nous avons vu en effet, les difficultés que causaient les options USAGE, PLACE qui sont ambiguës. Elles ne pourraient être définies de façon rigoureuse par la métalangue ALGOL.

C'est pourquoi, nous nous sommes limités à la division TRAITEMENT et à quelques extensions pratiquerent toujours compatibles, et qui vont fournir les seuls champs de comparaison possibles entre ALGOL et COBOL.

5. COMPARAISON ENTRE ALGOL ET LA DIVISION TRAITEMENT DE COBOL.

5. 1. Eléments des langages.

5. 1. 1. Symboles de base.

On ne peut comparer les symboles de base ALGOL et COBOL, car ces derniers sont beaucoup plus nombreux, puisqu'il existe des symboles pour les opérations d'entrée-sortie, le verbe EXAMINER etc... qui n'ont pas leurs équivalents en ALGOL. Mais même dans les cas où la correspondance sera possible comme dans les expressions arithmétiques simples, COBOL autorise non seulement les opérateurs arithmétiques usuels mais encore leur écriture "en clair". Aussi dans ce premier paragraphe, nous nous limiterons aux jeux de caractères, en soulignant que :

. Les lettres minuscules autorisées en ALGOL ne le sont

pas en COBOL.

. Dans les opérateurs arithmétiques, ALGOL emploie le signe x pour la multiplication et le signe ↑ pour la puissance, alors que COBOL utilise respectivement * et ** .

. Dans les opérateurs logiques, COBOL utilise les symboles en clair.

. Dans les opérateurs de relation COBOL utilise plutôt les symboles en clair.

. Dans les séparateurs, les symboles ; : et := n'existent pas en COBOL.

. Les règles d'édition n'existant pas en ALGOL, le signe $ n'appartient pas à son alphabet (ni d'ailleurs à l'alphabet de la division TRAITEMENT de COBOL).

. Les crochets [ ] n'existent pas en COBOL.

5. 1. 2. Représentation des nombres :

Le facteur de cadrage (symbole$_{10}$) n'existe pas en COBOL. Les autres règles d'écriture sont identiques.

5. 1. 3. Identificateurs de variable.

Un identificateur de variable en ALGOL commence toujours par une lettre. En COBOL, cet identificateur doit comporter au moins une lettre mais pas nécessairement la première. De plus cet identificateur peut comporter un trait d'union, ce qui est interdit en ALGOL. Un identificateur ne peut comporter plus de 30 caractères en COBOL, alors qu'en ALGOL, sa longueur est théoriquement illimitée mais pratiquement réduite en général, à quelque chose d'inférieur à 12 caractères.

Dans les variables indiciées, le nombre d'indices limité à 3 en COBOL est sans limitation dans ALGOL, et dans ce dernier cas ils peuvent être des expressions arithmétiques alors qu'en COBOL, ce ne sont que des variables simples ou entiers sans signe.

Les crochets d'indices [et] sont spécifiques à ALGOL alors que COBOL utilise des parenthèses normales (pas de risque de confusion d'après la structure même des indices).

La structure de fichier et la hiérarchie par niveaux de données d'une part, la structure de bloc d'autre part, font que la qualification est unique à COBOL.

5. 1. 4. Chaines et libellés non-numériques.

Les chaines en ALGOL ne peuvent apparaître que comme paramètres effectifs de procédure. En COBOL, leur équivalent c'est à dire les libellés alphanumériques peuvent servir dans les instructions de transfert et les affectations de valeur dans les sections INTERMEDIAIRES et CONSTANTES de la division DONNEES. Les constantes figuratives n'existent pas en ALGOL.

5. 2. Expressions arithmétiques et formules.

COBOL n'autorise pas la division entière (symbole ALGOL ÷). Les priorités des opérateurs arithmétiques sont les mêmes dans les deux langages. Les primaires arithmétiques diffèrent en ce sens que les indicateurs de fonction (même standard) n'existent pas en COBOL. A cette exception près, les formules COBOL et les expressions arithmétiques simples ALGOL sont équivalentes. Il n'y a pas d'expressions arithmétiques autres que simples en COBOL.

5. 3. Instructions d'affectation.

L'affectation simple d'une valeur numérique se fait en COBOL, dans les sections INTERMEDIAIRES et CONSTANTES de la division DONNEES par l'option VALEUR, ou dans la division TRAITEMENT par l'instruction TRANSFERER... EN... Dans ce dernier cas, la liste des parties gauches décrites en ALGOL devient la liste des noms de données (cf III. 3. 6. 5.). D'autre part ces affectations de valeurs peuvent se faire par option résultat (cf III. 3. 6. 3.) dans la cas d'instructions arithmétiques autres que CALCULER et par l'opération d'assignation dans ce cas.

Ainsi Z := X+Y+Z; et AJOUTER X, Y, Z. sont équivalents.

Il est à noter que le test DC ainsi que l'opération ARRONDI n'existent pas en ALGOL.

5. 4. Etiquette. Instruction ALLER A. Expression de de destination.

Les étiquettes se reconnaissent en ALGOL, lorsqu'elles servent de repères au début d'une instruction parce qu'elles sont suivies du symbole :.L'équivalent en COBOL est réalisé par le format de référence (début de l'étiquette en colonne 8 et suivie d'un.). L'identificateur d'aiguillage n'existe pas en COBOL mais l'option ALLER... SELON... et l'instruction CHANGER jouent ce rôle avec comme différence que seule une variable simple peut-être employée comme identificateur d'aiguillage et non une expression arithmétique.

Ainsi ALLER A AIG N ; P1:...;P2:...;P3:..; écriture ALGOL

et ALLER A P1, P2, P3 SELON N. P1... . P2. ... P3. ... écriture COBOL ont la même signification.

Les expressions de désignation d'ALGOL n'ont pas leur équivalent en COBOL.

5. 5. Expressions booléennes.

Les variables booléennes n'existent pas en COBOL mais on peut dire que les noms de condition et noms spéciaux sont des variables booléennes déguisées. En effet écrire dans la division DONNEES

03 VILLE; LONGUEUR 1.
88 PARIS; VALEUR 4.

puis dans la division TRAITEMENT
SI PARIS ALORS
revient à écrire dans la notation ALGOL :

ENTIER VILLE ; BOOLEEN PARIS ; PARIS := VILLE = 4 ; SI PARIS ALORS...

Paris devient bien alors une variable booléenne.

Les primaires booléens de COBOL ne pourront donc être une variable ou une valeur logique mais un nom de condition ou un nom spécial, et comme dans ALGOL une relation, les relations ayant les mêmes définitions, avec en plus la possibilité d'avoir des li-

bellés alphanumériques comme sujet ou objet de la relation. Bien entendu, il ne peut y avoir d'indicateur de fonction en COBOL, comme il ne peut y avoir de test de classe en ALGOL.

COBOL n'autorise pas les opérations logiques $\supset$ et $\equiv$. Pour les trois autres l'ordre de priorité est le même qu'en ALGOL. L'expression booléenne de COBOL correspond donc au terme booléen de ALGOL.

5. 6. Programmation des boucles.

En ALGOL, lorsqu'on veut répéter une instruction, on écrit par la proposition POUR le nombre de fois ù cette instruction doit être répétée puis on écrit cette instruction. En COBOL, on EXECUTE un certain nombre de fois une instruction écrite à un autre endroit du programme. Ce que l'on peut donc comparer est la limitation du nombre de boucles.

La première forme de l'instruction POUR en ALGOL n'a pas son équivalent COBOL. Les 2ème et 3ème forme de l'instruction POUR et la liste d'exécution primaire de l'instruction EXECUTER (cf III 3. 6. 4.) sont équivalentes, avec comme différence cependant que les valeurs initiales et les pas ALGOL autorisent les expressions arithmétiques alors qu'en COBOL seuls les opérandes sont admis, et qu'il ne peut y avoir de listes de POUR.

Comme instructions suivant la proposition POUR, ALGOL autorise une nouvelle instruction POUR qui elle même peut comporter une instruction POUR, et il n'y a pas de limite à cette récursivité Ceci ne peut se faire qu'avec trois niveaux maximum en COBOL.

5. 7. Blocs et instructions.

La notion de blocs imbriqués n'existe pas en COBOL, le programme COBOL ne formant qu'un seul bloc local au sens ALGOL. Par contre, on peut considérer les paragraphes et instructions comme des instructions composées, les symboles DEBUT et FIN étant remplacés par des points.

On ne peut alors donner des définitions semblables des instructions conditionnelles et inconditionnelles des deux langa-

gages. En effet dans l'instruction SI ALGOL, l'instruction suivant ALORS doit être inconditionnelle alors qu'en COBOL, elle peut être conditionnelle. Cette difficulté est levée en ALGOL par l'intermédiaire d'une instruction composée qui elle peut être conditionnelle.

Ainsi l'organigramme de la page suivante sera traité:

En COBOL de la façon suivante

SI C1 ALORS SI C2 ALORS I1 AUSSI I2 SINON I3 SINON SI C3 ALORS I4 SINON I5.

et en ALGOL

SI C1 ALORS DEBUT SI C2 ALORS DEBUT I1; I2 FIN SINON I3 FIN SINON SI C3 ALORS I4 SINON I5;

Les règles de succession et d'emplois d'instructions finales (ou ALLER A) sont identiques de même que la suppression de SINON PHRASE SUIVANTE (ou SINON ALLER A) et remplacement par . (ou ;).

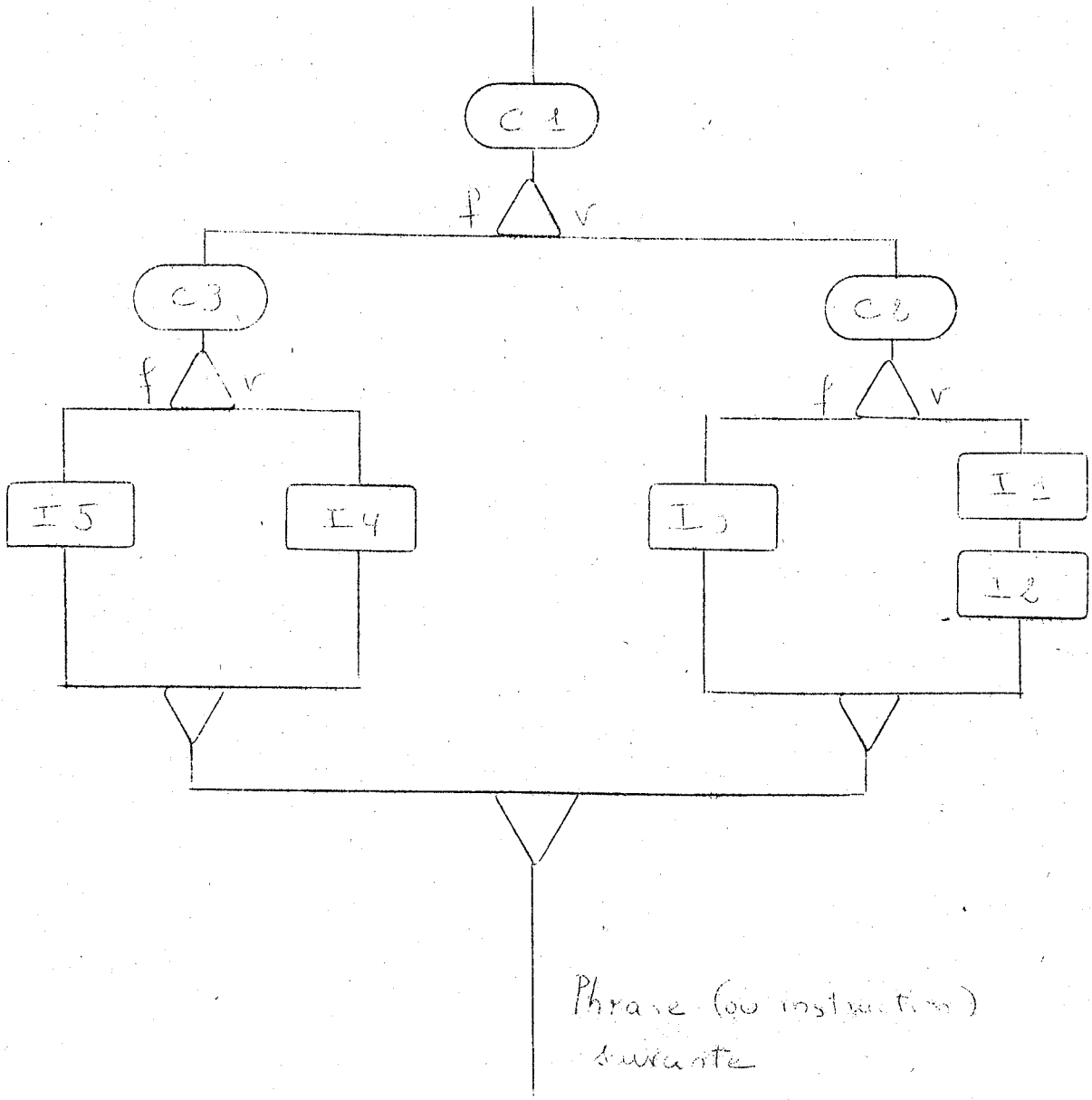
5.8. Déclarations.

Nous ne comparerons pas la façon de déclarer les variables ,puisque la structure de fichier n'existe pas en ALGOL, mais les types de variables pouvant être déclarées.

Les variables réelles et entières existent dans les deux langages et nous avons vu ce qu'il fallait penser des variables booléennes en COBOL. En COBOL, la structure de bloc n'existant pas, toutes les variables sont locales et il n'y a pas de variable de type REMANENT (sauf bien entendu les constantes qui ne peuvent être réellement considérées comme des variables).

Par contre COBOL peut déclarer le MODELE d'une variable alors qu'une procédure devra être écrite par ALGOL

Dans les déclarations de tableaux, les paires de bornes de COBOL sont nécessairement des variables simples, pouvant d'ail-


C 1
f
v
C3
C2
f
v
f
v
I 5
I 4
I 3
I 1
I 2
Phrase (ou instruction)
suivante

leurs varier, tandis qu'en ALGOL, ce sont des expressions arithmétiques.

Les déclarations d'aiguillage n'existent pas en COBOL.

5. 9. Procédures.

Nous avons vu que COBOL, n'utilisait pas de fonction standard. Par contre, il existe des procédures d'Entrée-sortie qui manquent à ALGOL. Les indicateurs de fonction n'existent pas en COBOL, mais on peut réaliser des instructions procédures par l'intermédiaire des verbes DEFINIR et INCLURE. Comme dans ALGOL, le corps de procédure pourra être une instruction programmée en langage source ou du code par l'intermédiaire du verbe ENTRER.

Ainsi la procédure SOMCAR s'écrirait en ALGOL :

PROCEDURE SOMCAR (X, Y, U) ; U := X ↑ 2 + Y ↑ 2 ;

et en COBOL

DEFINIR SOMCAR FORME SOMCAR (X, Y, U).

CALCULER U = X ** 2 + Y ** 2.

et plus tard sera employé de la même façon qu'en ALGOL mais avec appel par VALEUR uniquement.

Si cette procédure est une procédure en bibliothèque, on pourra l'appeler de la façon suivante :

INCLURE SOMCAR REMPLACANT X PAR..., Y PAR..., U PAR... .

les PAR étant suivis de noms de donnée ou de libellés jouant le rôle de paramètres effectifs.

Nous avons vu que les procédures d'entrée-sortie étaient standard, mais on peut en fabriquer d'autres grâce au verbe UTILISER ce qui ne restreint donc pas la généralité.

Il n'y a pas de procédures récursives en COBOL.

6. CONCLUSION.

Ces deux langages ont beaucoup de points communs, mais leurs différences essentielles (blocs ALGOL, divisions COBOL)

semblent difficilement surmontables pour tenter de les réunir en un seul langage. Mais au niveau même des applications, la structure de fichiers n'existant pas en ALGOL, on voit mal comment il pourrait traiter des problèmes de gestion. D'autre part COBOL ne présente pas assez de formalisme pour s'appliquer à des algorithmes mathématiques. On pourra bien sur trouver des exemples simples où l'un et l'autre langage peut être utilisé mais cela est sans aucune portée pratique.

En fait, le langage universel, synthèse d'ALGOL et COBOL que quelques auteurs américains semblent préconiser, est sans intérêt et il est plus raisonnable de perfectionner ces langages ou leurs traducteurs, chacun dans leur domaine propre afin d'en faire des langages standard de programmation.

On peut remarquer également qu'aucun de ces deux langages n'est d'utilisation facile pour les problèmes de recherche opérationnelle et un troisième langage orienté pourrait être défini dans ce domaine.

°°°

IV. QUELQUES DONNEES PRATIQUES SUR LA COMPILATION DE COBOL SUR UNE MACHINE A MOTS.

1. DEFINITION DU LANGAGE SOURCE A COMPILER.

Avant de donner des aperçus sur une méthode de compilation, il faut nécessairement restreindre le problème, c'est-à-dire d'une part bien spécifier le genre de machine sur laquelle et pour laquelle la compilation se fera, mais encore d'autre part définir le langage source qui sera compilé.

Les auteurs du rapport COBOL ont, dans leur définition du langage, laissé la voie ouverte à ces deux restrictions puisque la première fait l'objet du paragraphe CONFIGURATION de la division EQUIPEMENT et la deuxième est sous-entendue par la présence - dans le rapport même - d'options facultatives pour un COBOL-restreint et possibles pour un COBOL-étendu.

Ce ne sont pas les restrictions apportées à la division TRAITEMENT qui influeront sur la compilation, car ces restrictions ne feront que supprimer quelques sous-programmes dans la partie génération du compilateur le plus général. Par contre, la capacité de la mémoire centrale, le nombre de mémoires auxiliaires (tambours, bandes magnétiques) et la structure interne de la forme de traitement de l'information seront d'une grande importance, non seulement dans la méthode de compilation, mais également comme nous l'avons déjà vu, sur la définition du langage source.

Pour ce dernier, la capacité de la mémoire rapide donnera des limites pour la segmentation, le nombre de mémoires auxiliaires et d'organes d'entrée-sortie apportant des restrictions en général sans importance pour une installation déterminée qui doit posséder un matériel convenant aux genres de problèmes qu'elle a à effectuer, sur le nombre de fichiers pouvant être traités simultanément. Quant à la structure interne elle fournira tous les renseignements supplémentaires sur les clauses CLASSE, USAGE, sur les positionnements de mots dans les mémoires ainsi que sur les trai-

tements d'entrée-sortie.

Pour la compilation, les dimensions physiques du calculateur influeront sur le nombre de passages nécessaires pour réaliser la traduction du langage. La structure interne pourra modifier l'adressage symbolique des mots, le genre de codification et la forme des instructions générées. Ceci est particulièrement évident dans le cas de machines à mots ou à caractères puisque dans le premier cas il faudra traduire l'information de longueur donnée en caractères par une information d'adresse susceptible d'être comprise par le calculateur alors que dans le second cas il suffirait d'un compteur.

La suite de ce chapitre s'appliquera à une machine à mots de capacité moyenne, le GAMMA-60, (mémoire centrale 16 384 mots dont 4 096 occupés par la gestion générale, pas de tambour, autant de bandes qu'il en sera nécessaire pour la compilation) et dont le mot machine (ou catène) comprend 24 bits. Cette machine est d'ailleurs peu adaptée à COBOL car la longueur - en forme interne - d'un caractère alphanumérique est 6 bits tandis que celle d'un caractère numérique est 4 bits. De plus les quantités sur lesquelles on veut réaliser des opérations arithmétiques doivent être en flottant sur 2 catènes ce qui nous amènera à définir un grand nombre d'USAGES (cf IV.4. 5. 1.) ainsi qu'à générer des instructions de "splittage" de catènes ou de conversions d'une forme ou d'une autre.

Ainsi l'instruction COBOL :
AJOUTER X A Y.

où X a été défini sous forme INTERNE-1 (flottant sur deux catènes)
Y a été défini sous forme EXTERNE-1 (4 bits/caractère)

devra être généré de la façon suivante :

a) Conversion de Y sous forme flottante soit Y'

b) X + Y' = Y'

c) Conversion de Y' sous forme EXTERNE-1 en Y' (en optimisant les réservations de mémoire).

d) Transfert de Y' en Y.

Nous indiquerons dans la suite les principales restrictions qui ont été apportées au langage source à l'endroit où ces restrictions apparaissent pour la première fois.

2. ORGANISATION GENERALE DE LA COMPILATION.

En général, la compilation d'un langage, c'est-à-dire la traduction d'un langage source en un langage objet, doit passer par un stade intermédiaire, ce que l'on peut schématiser ainsi :

1°. Traduction du langage source en un langage intermédiaire ;

2°. Traduction de ce langage intermédiaire en langage objet.

Ici ce langage objet doit être un langage d'exécution pour le calculateur, soit langage machine, soit langage d'assemblage. Nous nous limiterons à ce dernier cas, ce qui revient simplement à ne pas effectuer une traduction supplémentaire.

La première partie de la compilation peut présenter des formes diverses, soit chaine codée d'unités syntaxiques de même longueur, soit suite d'états syntaxiques correspondant à des instructions machines préalablement bien définies, soit pseudo-instructions associées à des listes de paramètres etc...

La deuxième partie traitera le résultat de la première et il en découlera le langage objet, généralement langage d'assemblage.

Certains traducteurs de langage scientifique réalisent ces deux opérations simultanément, d'autres utilisent 2 (et quelquefois plus) passages machine bien distincts. |18|. De par sa nature volumineuse, un traducteur COBOL nécessitera un nombre important de passages. A titre d'exemple des compilateurs existant sur 1401 |16| ou GAMMA-30 |17| demandent une quarantaine de passages. Ceci tient principalement au fait que COBOL étant peu formel -dans son écriture - un nombre important de phases sera nécessaire avant d'arriver au langage intermédiaire.

Grossièrement, on peut diviser la traduction de COBOL en quatre parties :

1. Extension et premières vérifications sur le langage source.

2. Formalisation et précodification.

3. Codification et adressage symbolique des données. Extension d'instructions de la division TRAITEMENT.

4. Génération d'instructions du langage objet.

Cette division en 4 parties ne veut pas dire que 4 phases ou quatre passages machine seront suffisants mais indique seulement les grandes lignes de la traduction.

3. EXTENSIONS ET PREMIERES VERIFICATIONS SUR LE LANGAGE SOURCE.

3. 1. Tri sur n° séquentiel.

Dans le cas où le programmeur a inscrit le n° séquentiel dans les colonnes 1 à 6 du format de référence, un tri suivi d'un interclassement seront faits sur ces numéros de façon à respecter l'ordre du programme source au cas où il y aurait eu des erreurs. Ceci peut se faire au moment du passage carte-ruban. Si ce n° séquentiel n'existe pas - ce que l'on pourrait savoir par l'introduction d'une carte paramètre en tête du programme source - on aurait une simple conversion carte-ruban.

3. 2. Options COPIER.

Il faut distinguer ici les options COPIER DEPUIS BIBLIOTHEQUE des options COPIER se rapportant à d'autres informations du programme source. Pour ce qui est de la bibliothèque, on peut supposer qu'une écriture par blocs copiables directement rendrait assez facile ce passage. Il est cependant à noter que pour la division DONNEES, cela pose de nombreux problèmes si l'on autorise la copie de mots de n° de niveaux différents à cause des réajustements de ces n°.

Pour l'autre option COPIER, il faut envisager deux passages, l'un produisant une table des descriptions de fichier et

d'articles à copier (avec ajustement des numéros de niveau si besoin est) ; l'autre faisant l'opération COPIER elle-même c'est-à-dire un interclassement.

Ces opérations qui peuvent paraître simples sont en fait compliquées par la détermination du mot COPIER, ceci surtout dans la division DONNEES, et la détermination ensuite du mot à COPIER et de l'étendue à copier.

Dans un premier compilateur on pourrait se limiter aux options COPIE DE de la division EQUIPEMENT et COPIER dans la division DONNEES portant uniquement sur des articles courants ce qui réduirait de beaucoup les difficultés signalées.

4. FORMALISATION ET PRECODIFICATION.

Le but général de cette partie du compilateur est de se débarasser de la structure lourde et redondante du langage source en effectuant les opérations suivantes.

Lors du traitement des divisions EQUIPEMENT et DONNEES, on affectera à chaque identificateur une précodification qui servira dans la suite à déterminer l'adressage symbolique des données ainsi qu'un équivalent lexicographique de longueur 1 catène afin d'avoir à traiter des noms d'identificateur de longueur fixe et hiérarchisée, ce qui facilitera de nombreuses consultations de table.

Lors du traitement de la division TRAITEMENT, on remplacera chaque mot clé par un catène à codification précisée dans la suite et chaque identificateur par un mot machine définissant son genre (étiquette, libellé, nom de donnée). Une table de ces identificateurs sera formée puis fusionnée avec les tables correspondantes s'il y a lieu.

Un certain nombre d'erreurs syntaxiques, sur le jeu de caractères autorisés, la longueur des identificateurs, le format de référence seront détectables dans cette partie. Nous citerons les principales au fur et à mesure de leurs occurences possibles.

Nous allons maintenant détailler certaines des opérations qui se font lors de cette partie du compilateur. Nous ne les indiquerons pas dans l'ordre réel dans lequel elles apparaissent au moment de la compilation, mais dans un ordre logique établissant les liaisons entre chacune de ces opérations.

4.1. Détection des mots du langage source et vérifications du format de référence. Premières erreurs syntaxiques.

La fin d'un mot se détecte en général par un espace, mais l'existence de la colonne 7 du format de référence et son rôle particulier entraine certaines difficultés. Le programme de détection de mot est spécifique à chaque division car il existe dans la division TRAITEMENT des difficultés supplémentaires telles par exemple les trois significations du point (point de fin de phrase, point suivant une étiquette, point décimal), les deux significations des parenthèses (parenthèses arithmétiques, crochets d'indice); les signes unitaires etc...

La méthode générale consiste à éclater l'image de la carte, c'est-à-dire à ranger chaque caractère représenté par une colonne de la carte (ou du format de référence) dans une mémoire, ce caractère étant cadré à droite, et à ranger dans une mémoire compteur le n° de la colonne ceci afin de tester les colonnes 7, 8, 12, 73 du format de référence qui jouent un rôle particulier. Puis ces caractères sont regroupés dans une zone de manoeuvre jusqu'à ce que l'on ait détecté un caractère non espace après un (ou plusieurs) espace. Lorsque ceci a été fait, on se renvoie sur un sous-programme dont l'adresse a été définie grace au traitement du mot précédent.

Pour illustrer ceci, prenons l'exemple très simple suivant tiré de la division TRAITEMENT.

```
78   12
 PARAG. MULTIPLIER  X  DANS  Y  PAR  Z.  ...
```

On suppose que la ligne précédente s'était terminée par un point.

Le programme principal détermine la fin du mot. Ici après rencontre d'un point fin de phrase, il testera la colonne 8. Comme la colonne 8 ne comprend pas un espace, la mémoire ADRESSE comportant l'adresse du S. P. de traitement à exécuter contiendra l'adresse du S. P. de traitement des étiquettes. Après avoir détecté la fin du mot PARAG grâce à un espace suivi de M , on se renvoie sur le S. P. traitement d'étiquettes qui placera dans la mémoire ADRESSE l'adresse du S. P. de consultation de tables des verbes, car l'on sait qu'après un point étiquette on a nécessairement un verbe ou un assimilé. Après reconnaissance de MULTIPLIER, ADRESSE contiendra le S. P. affecté aux opérandes, puisqu'un opérande suit nécessairement MULTIPLIER et on rangera dans une deuxième mémoire ADRSUI l'adresse du S. P. de reonnaissance des mots clés pouvant suivre le dernier mot clé rencontré, ici nécessairement le S. P. de reconnaissance de PAR. Puis on détermine X, et on se renvoie sur le S. P. traitant les opérandes dont l'adresse est dans ADRESSE Le processus se répète ainsi ce qui permet de détecter un grand nombre d'erreurs sur les mots clés manquants ou mal placés.

Ce même programme principal détecte également les erreurs du format de référence telles des débuts de mots en colonne 9-10-11 des mots de longueur supérieure à 30 caractères etc...

4. 2. Précodification de la division IDENTIFICATION.

La seule opération est la reconnaissance de la présence du paragraphe PROGRAMME.

4. 3. Précodification de la division EQUIPEMENT.

Nous nous sommes restreints à une simple reconnaissance de la section CONFI URATION. Le paragraphe NOMS-SPECIAUX pourrait être traité dans une seconde version plus approfondie du compilateur d'une façon ressemblant d'assez près à celle des équivalents lexicographiques de fichier.

La précodification de la section ENTREE-SORTIE se compose de trois tables :

1. Une table FICLEX (fichier-lexicographe) qui affectera un équivalent lexicographique d'un catène à un nom de fichier.

2. Une table PCODFIC (précodification - fichier) qui permettra de stocker les premiers renseignements obtenus sur les fichiers.

3. Une table de REPRISE dont nous ne pouvons donner actuellement la structure, celle-ci dépendant de la gestion générale GAMMA-60 appliquée à COBOL qui n'est pas encore définitive.

4. 3. 1. La table FICLEX.

Comme les tables NDDLEX et ETIQLEX que nous formerons dans la suite, cette table a pour but de donner à chaque identificateur COBOL (ici les noms de fichier) un équivalent lexicographique unique de longueur fixe. Nous verrons plus en détail (cf IV, 4. 4.) les intérêts que cela présente.

L'équivalent lexicographique, sous forme interne alphanumérique ceci afin de faciliter l'édition, sera une lettre suivie de trois zéros. La table FICLEX se présentera comme une succession de groupements de catènes, chaque groupement possédant :

. 1 catène définissant la longueur du groupement ;

. De 1 à 8 catènes (30 caractères équivalent à 7 catènes et demi) donnant le nom en clair du fichier.

. 1 catène lexicographique de la forme A000.

La lettre C sera réservée aux constantes.
La lettre E sera réservée aux étiquettes.
La lettre M sera réservée aux mémoires de manoeuvre.

4. 3. 2. La table PCODFIC.

Cette table, formée par groupements de quatre catènes par fichier, rassemble tous les éléments d'information que l'on peut tirer sur la structure des fichiers par l'analyse de la section ENTREE-SORTIE.

Le premier catène servant d'indicatif au groupement est l'équivalent lexicographique du fichier considéré soit FEQLEX.

Le deuxième, soit PCODF, représente une codification du

paragraphe GESTION-FICHIERS, codification qui est la suivante :

bit 24 en position 1 si option EVENTUEL
bits 21-20-19 donnant le milieu externe soit

*	000	2 dérouleurs
	010	1 dérouleur
	011	1 dérouleur plusieurs bobines
	100	imprimante
	110	perforateur de cartes
	111	lecteur de cartes

bit 18 en position 1 indique qu'il n'y a pas de bascule dans le cas de traitement de fichier sur bande (les opérations d'E-S sur cartes et imprimante seront limitées aux traitements standard).

Le troisième catène donne un nom à la zone de lecture ou d'écriture, soit FZES.

Le quatrième, soit FZA, donne un nom à la zone de travail de l'article, en général A + (la signification du + sera évidente lors de l'adressage symbolique des mots de l'article), A étant la même lettre que celle de l'équivalent lexicographique de fichier, l'exception possible étant la présence de l'option MEME ZONE ARTICLE.

4. 3. 3. Edition et détection d'erreur.

Après ce passage, on éditera la division EQUIPEMENT avec à la fin de chaque ligne du listing du programme source, où existe un verbe PRENDRE, la catène FEQLEX du fichier attaché à ce verbe PRENDRE.

Les principales erreurs que l'on peut détecter sont :

- Les omissions de paragraphes ou de mots clés ;
- Des allocations de milieu externe inconnu.
- Une option MEME ZONE appliquée à des fichiers inconnus.

4. 4. Formation et recherche d'équivalents lexicographiques aux noms de données COBOL.

Nous quittons ici l'ordre séquentiel de la compilation

afin de présenter un traitement d'ensemble du lexicographe.

On veut donner à chaque nom de donnée COBOL un équivalent lexicographique unique de longueur fixe. Ceci présente les intérêts suivants :

. Traiter des identificateurs de longueur fixe et minimum dans la suite.

. Permettre une recherche systématique et rapide de la codification et de l'adressage symbolique de chaque identificateur.

. Détecter les erreurs sur les noms d'identificateurs d'une façon automatique.

L'inconvénient est que l'on a à former une table supplémentaire.

4. 4. 1. Structure de la table NDDLEX.

On part de :

. La division DONNEES telle qu'elle est écrite sur le format de référence.

. La table FICLEX (cf IV. 4. 3. 1.)

On aboutit à :

. La table équivalence NDDLEX (nom de donnée lexicographe) qui comprend pour chaque identificateur :

- 1 catène indiquant la longueur en catènes du nom de l'identificateur ;
- De un à huit catènes contenant le nom de l'identificateur
- Un catène contenant l'équivalent lexicographique EQLEX ;
- Un catène contenant l'adresse relative dans la table où trouver le prochain identificateur ayant la même initiale (ceci afin d'optimiser les recherches dans cette table), soit ALPHASUI ;
- Un catène indiquant la portée dans la table de cet identificateur soit PORTEE.

. La table TBALSUI (table alphabétique du suivant) qui donne l'adresse relative dans NDDLEX du premier identificateur commençant par une initiale donnée. Elle comprendra 36 catènes correspondant aux 26 lettres et 10 chiffres.

Illustrons ceci par l'exemple suivant d'une descriptio de fichier (sans les déclarations) et les tables NDDLEX et TBALSUI correspondantes. On suppose que l'adresse initiale de NDDLEX est 1.

	Longueur (1)	Nom (1 à 8)	EQLEX (1)	ALPHASUI (1)	PORTEE (1)	
FP	1	FP	A000	0	76	5
PERMANENT-1	3	PERMANENT-1	A001	59	57	12
DATE	1	DATE	A002	0	36	17
ANNEE	2	ANNEE	A003	0	10	23
X	1	X	A004	12	0	28
Y	1	Y	A005	0	0	33
MOIS	1	MOIS	A006	17	10	38
XI	1	XI	A007	2	0	43
X2	1	X2	A008	0	0	48
JOUR	1	JOUR	A009	0	0	53
MENSUEL	2	MENSUEL	A010	2	10	59
MOIS	1	MOIS	A011	0	0	64
T	1	T	A012	0	0	69
PERMANENT-2	3	PERMANENT-2	A013	0	5	76
Z	1	Z	A014	0	0	81

La dernière colonne de droite donne l'adresse, par rapport au premier catène supposé d'adresse 1, du catène PORTEE.

4. 4. 2. Structure de la table TBALSUI.

Formée de trente-six catènes, elle aurait la forme suivante dans l'exemple ci-dessus :

A)18	(B)0	(C)0	(D)13	(E)0	(F)1	(G)0	(H)0	(I)0	(J)49	(K)0	(L)0
M)34	(N)0	(O)0	(P)6	(Q)0	(R)0	(S)0	(T)5	(U)0	(V)0	(W)0	(X)24
Y)29	(Z)77	(Ø)0	(1)0	(2)0	(3)0	(4)0	(5)0	(6)0	(7)0	(8)0	(9)0

La lettre ou le chiffre entre parenthèses n'étant là évidemment que pour la compréhension et n'existant pas dans la tab réelle.

4. 4. 3. Formation de ces tables.

Pour former NDDLEX, on se servira de deux stacks de cinquante et un catènes chacun (ce nombre correspondant aux nombres de niveaux possibles). Le premier de ces stacks conservera l'adresse des différents catènes PORTEE, et le deuxième la longueur de ses portées. Les adresses de ces stacks sont indiciés par rapport à leur origine, par un catène NN conservant le n° de niveau du dernier identificateur traité et accessoirement par un catène NN1 dont le contenu est celui de NN-1.

Lorsqu'on rencontre un identificateur de n° $\geqslant$ NN , on forme NDDLEX, c'est-à-dire le catène longueur, les catènes NOMS, EQLEX, et on laisse deux catènes vierges. On stacke l'adresse du dernier, c'est-à-dire du catène PORTEE et la longueur de ce qu'on vient d'écrire est accumulée dans le deuxième stack.

Lorsqu'on rencontre un identificateurrde n° $<$ NN, on range la longueur de la portée (2ème stack) à l'adresse délivrée par le premier et on fait regresser ces deux stacks.

Les cas particuliers où l'on saute des niveaux sont résolus grâce au catène NN1.

Illustrons ceci par l'exemple précédent

	adresse						portée							
Id.	00	01	02	03	04	05	00	01	02	03	04	05	NN	NN1
FP	5						0						0	
PERMANENT-1		12						7					1	
DATE			17						5				2	
ANNEE				23						6			3	
X					28						5		4	
Y					33					16	10		4	
MOIS				38						21			3	
X1						43						5	5	
X2						48				31		10	5	4
JOUR				53					41	36			3 - 2	2
MENSUEL			59						47				2	
MOIS				64						5			3	
T				69				64	57	10			3 -- 2	2 -- 1
PERMANENT-2			76					71					3-- 2-- 1	2 -- 1
Z				81			76	76	5				2 -- 1	1 --0

Pour former TBALSUI et ALPHASUI, on associe à chacun des catènes de la table, donc à chaque initiale possible, un nouveau catène qui contiendra l'adresse dans NDDLEX du catène ALPHASUI correspondant.

Ainsi lorsqu'on rencontre FP, on met dans TBALSUI(F) = TBALSUI + 5 l'adresse dans NDDLEX du catène longueur de FP, ici l'adresse début de NDDLEX soit 1, et dans catène associé l'adresse du catène ALPHASUI de FP. A la prochaine occurence d'un identificateur commençant par F, on mettra l'adresse dans NDDLEX de ce nouvel identificateur commençant par F, on mettra l'adresse dans NDDLEX de ce nouvel identificateur à l'adresse indiquée par le catène associé.

4. 4. 4. Structure de la table formée à partir de la division TRAITEMENT.

Lorsqu'on rencontre un nom de donnée dans la division TRAITEMENT, on le range dans la table TRNND (traitement nom de donnée) de la façon suivante :

- Il n'y a pas qualification. On range la longueur en catènes suivie du nom en clair.
- Il y a qualification. Chaque catène portant longueur du mot qualifié aura un 1 en bit 24.

Ainsi dans l'instruction AJOUTER X, X2 DANS MOIS DANS DATE DANS PERMANENT-1 DANS FP.

On aura à l'endroit équivalent de TRNND :

I	X	I I	X2	I I	MOIS	I I	DATE	I 3	PERMANENT-I	I	FP

La formation de cette table n'offre aucune difficulté.

4. 4. 5. Fusion des tables NDDLEX et TRNND.

On veut former une table qui donne la correspondance lexicographique des identificateurs de TRNND. Dans l'exemple précédent on aurait :

A004|A008|

Ici encore deux cas se présentent :

Pas de qualification (bit 24 = 0 dans le catène longueur de TRNND). On détecte dans TBALSUI l'adresse du premier identificateur commençant par la même initiale. On teste l'égalité de TRNND et NDDLEX d'abord par le catène longueur puis par les catènes NOMS. En cas de différence on se renvoie dans NDDLEX à l'identificateur dont l'adresse est contenue dans ALPHASUI. En cas d'égalité on détermine l'EQLEX correspondant qu'on range dans FUNND (fusion des noms de données).

. Il y a qualification. On cherche le qualificateur de plus haut niveau (en stackant les adresses dans TRNND des qualifiés, dans un stack de cinquante mémoires maximum). A l'aide des qualificateurs, on détermine deux adresses, l'une donnant l'adress minimum dans NDDLEX du début possible des adresses des qualifiés inférieurs, l'autre donnant, grâce à PORTEE, l'adresse maximum. On réalise ces opérations jusqu'à la détermination du nom de donnée de niveau inférieur c'est-à-dire jusqu'à ce que le stack d'adresse soit vide. On est alors ramené à trouvé un EQLEX avec des bornes pour les adresses dans NDDLEX.

Toutes ces opérations permettent de déterminer doux erreurs principales dans le langage source :

- On n'a pas trouvé dans NDDLEX un identificateur correspondant à un pris dans TRNND, soit parce que les orthographes diffèrent, soit parce que la qualification est erronnée.

- On trouve deux EQLEX possibles parce que la qualification est insuffisante.

4. 5. Précodification de la division DONNEES.

En même temps que l'on formera la table NDDLEX, une précodification succincte de la division DONNEES sera réalisée. Vu l'encombrement et la lourdeur du langage source, un passage sera uniquement affecté à formaliser et à enregister les éléments d'information que l'on peut obtenir par une lecture de cette division. Puis ces informations seront reprises afin d'obtenir toutes les indications nécessaires à l'adressage des données.

4. 5. 1. Restrictions apportées au langage source. Définition des différentes CLASSE et USAGE. Précisions sur l'option PLACE.

Au moment de traiter les descriptions de fichier, les seules informations reconnues par le compilateur seront celles ayant trait aux REPERES, la longueur des BLOCS, et les NOMS DES ARTICLES ; Le traitement des repères sera spécifié suivant les repères standard BULL notamment en ce qui concerne la clause VALEUR DE ID. Les seules options possibles seront STANDARD et OMIS

Dans un premier compilateur, on ne traitera que des articles de longueur fixe. De plus la clause "LONGUEUR entier MOTS" pourra suivre la clause "BLOC entier ARTICLES" dans le cas où un fichier contient des articles de NOMS différents et de longueurs différentes.

La structure interne du GAMMA-60 appelle des définitions de CLASSE et d'USAGE particulières. On considérera les genres de mots suivants :

. AN sans édition dont les caractères seront représentés par six bits, ces caractères ne pouvant être supprimés ou traités séparément.

. AN avec édition et sans insertion. Caractères à six bits. Le MODELE du mot ne peut comporter les symboles O B , . CR $ + - (ces trois derniers pouvant exister s'ils se trouvent en-tête du MODELE.

. AN avec édition et insertion. Caractères à 6 bits. MODELE quelconque.

. ALPHABETIQUE ne comportant que des lettres.

. NUMERIQUE EXTERNE-1 Caractères à quatre bits. Le MODELE ne pourra comporter que les symboles S 9 V P

. NUMERIQUE EXTERNE-2 Caractères à six bits forme normalisée. Le fichier devant être sur cartes.

. NUMERIQUE EXTERNE-3 Caractères à six bits forme programmée.

. NUMERIQUE INTERNE-1 Mot sur un catène sous forme binaire.

. NUMERIQUE INTERNE-2 Mot en flottant sur deux catènes.

Les MODELES ne pourront porter que sur les formes EXTERNE

Les mots INTERNE seront automatiquement placés dans une ou deux catènes et ne pourront subir d'option CADRE ou PLACE.

Un même catène ne pourra comprendre des mots (ou parties de mots) ayant des représentations internes différentes, c'est-à-dire à six et quatre bits.

Lorsque l'USAGE d'un mot NUMERIQUE n'est pas spécifiée, le mot sera EXTERNE-1.

Dans le cas de tables ou de listes, le nombre de FOIS où un mot peut figurer sera fixe.

Les mots INTERNE sont algébriques. Il n'est pas besoin de le spécifier. Pour les mots NUMERIQUE EXTERNE-1 lorsque la clause ALGEBRIQUE est spécifiée une position supplémentaire (quatre bits) sera retenue et réservée au signe, celui-ci étant placé en tête.

L'option BITS dans la clause VIRGULE, les clauses ETENDUE et RECOUVRE ne seront pas implantées ; dans un premier compilateur, on n'autorise qu'une VALEUR par nom de condition.

4. 5. 2. <u>La table TBPCODO et ses tables satellites</u>.

La table TBPCODO (table précodification 0) va enregistrer les éléments d'information que l'on peut obtenir aisément en un premier passage. Elle est formée de groupements ayant la structure suivante :

- 1 catène indiquant la longueur du groupement ;
- 1 catène EQLEX identifiant le groupement.

Et accessoirement :

- 1 catène LG donnant la longueur en caractères du mot telle qu'elle est déclarée dans la rubrique de description du mot ;
- 1 catène PCODAUX dont nous verrons la structure ci-dessous (1)

Dans le cas de description de fichier, la table TBPCODO ne comportera que les deux premiers catènes décrits ci-dessus. La table PCODFIC (cf IV. 4. 3. 2) sera complétée par :

(1) - de 1 à 3 catènes reproduisant le MODELE dans le cas du choix de cette option

bit 17 en position 1 si REPERES OMIS
bit 16 en position 1 si longueur du BLOC est en mots.
bits 15 à 1 donnant la longueur du bloc.

Structure du catène PCODO.

Nous aurons la codification suivante :

. bit 24 en position 1 si option REDEFINIT
. bit 23 " " 1 " mot groupe ou article courant
. bit 22 " " 1 " mot élémentaire ou article courant
. bits 21 à 16 n° de niveau en binaire avec 01 = 000000
77 = 110100
88 = 111000
. bits 15 et 14 nombre d'indices affectés au mot
. bits 13 à 10 codification des clauses CADRE et PLACE
. bit 9 en position 1 si option valeur dans le cas d'INTERMEDIAIRES ou CONSTANTES
. bit 8 à 5 codification des CLASSF et USAGE ainsi qu'ALGEBRIQUE.
. bits 4 et 3 clauses VIRGULE
. bits 2 et 1 nombres de catènes suivant non compris ceux reproduisant le MODELE.

Structure du catène PCODAUX

Ce catène divisé en trois parties donne :

. 4 bits pour la position de la VIRGULE.
. 9 bits pour les clauses d'édition non traitées par MODELE.
. 11 bits pour le nombre de FOIS où figure un article.

La table TBMOD

Le traitement des MODELES pouvant apporter de nombreuses complications et vérifications, un passage spécial lui sera réservé. Aussi met-on en réserve dans une table les EQLEX des différents mots comportant des MODELES dans leur rubrique de description, et on écrit la reproduction du MODELE dans la table TBPCODO.

La table TBNCOND

Cette table comporte les EQLEX des différents noms de condition suivis des EQLEX des variables conditionnelles auxquels ils se rapportent et des EQLEX des libellés suivant les différentes clauses VALEUR.

La table TBVALEUR

Elle donne les valeurs des CONSTANTES ou INTERMEDIAIRES à valeurs initiales. Les transferts correspondants seront générés en début de programme objet.

4. 5. 3. Le traitement des MODELE.

Le rapport COBOL définit d'une façon légèrement ambigüe ; les successions autorisées de caractères dans les MODELES. Aussi avons nous défini deux matrices de succession |19|, la première indiquant quels sont les caractères ne pouvant suivre immédiatement un caractère donné, la seconde indiquant les successions impossibles jusqu'à la fin du MODELE. Nous avons concentré ces deux matrices en un simple tableau, les 0 étant communs aux deux matrices, les 1 hachurés indiquant des 1 communs aux deux matrices, et les 1 non hachurés étant ceux de la matrice de succession immé iate.

	A	B	CR	DB	P	S	V	X	Z	.	0	1 à 8	9	*	,	$	(	)	-	+	fin
dep	0	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0	1
A	0	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	0	1	1	1	0
B	1	0	0	0	1	1	0	1	1	0	0	1	0	1	0	0	0	1	0	0	0
CR	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
DB	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
P	1	0	0	0	0	1	0	1	1	1	0	1	0	1	1	0	0	1	0	0	0
S	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1
V	1	0	1	1	0	1	1	1	0	1	0	1	0	0	1	0	1	1	0	0	1
X	0	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	0	1	1	1	0
Z	1	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0
.	1	0	1	1	0	1	1	1	0	1	0	1	0	0	1	0	1	1	0	0	1
0	1	0	0	0	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0
1 à 8	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	0	1	1	1
9	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
*	1	0	0	0	0	1	0	1	1	0	0	1	0	0	0	0	0	1	0	0	0
,	1	0	1	1	1	1	1	1	0	1	0	1	0	0	1	0	1	1	1	1	1
$	1	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0
(	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1
)	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0
-	1	0	1	1	1	1	0	1	0	0	0	1	0	0	0	0	0	1	0	1	0
+	1	0	1	1	1	1	0	1	0	0	0	1	0	0	0	0	0	1	1	0	0

Ainsi un V peut suivre un S immédiatement, un S ne peut-être que le premier caractère, il ne peut y avoir de A après un S , une (ne peut précéder immédiatement une) etc...

On associera à chaque caractère deux catènes correspondant aux deux lignes de chaque matrice lui étant associées, ainsi qu'un test sur un bit, 24 pour A, 23 pour B etc... Après traitement de chaque caractère du MODELE on fera une réunion logique de ces catènes et le catène résultant servira de test de succession pour le caractère suivant. On minimise ainsi de beaucoup les aiguillages détectant les erreurs.

Le traitement du MODELE proprement dit servira à compléter les catènes PCODO et PCODAUX ainsi que le catène LG de TBPCODO. Au cas où il y aurait des insertions (ce qui sera déterminé dans PCODO par la codification de la CLASSE), une table TBMODIN donnera l'EQLEX du mot ainsi que le masque de sa configuration interne.

4. 5. 4. Edition de la division DONNEES.

A la fin de chaque ligne du listing d'une rubrique de description on indiquera l'EQLEX correspondant.

4. 6. Précodification de la division TRAITEMENT.

Cette partie du compilateur sert d'une part à transformer la division TRAITEMENT du langage source en une chaine codée d'unités syntaxiques de même longueur (1 catène) et à associer à ces unités syntaxiques des listes de paramètres (nom de donnée, libellés, étiquettes). Cette chaine codée sera formée grossièrement en un premier passage, puis améliorée dans la suite afin de se rapprocher de la chaine codée finale qui servira au moment de la génération.

4. 6. 1. Restrictions apportées au langage source.

Les opérandes des verbes arithmétiques ne pourront avoir que 10 chiffres (simple précision). Les verbes d'entrée-sortie ne pourront être précisés qu'au moment de la définition exacte

de la gestion BULL.

Lorsqu'un mot groupe est impliqué dans un transfert, une simple copie est réalisée. Pour les transferts mots-élémentaires

. Un message sera donné par le compilateur si la zone émettrice est plus longue que la zone réceptrice. Au cas où il n'y aurait ni correction ni cadrage spécial, on fera un tronçonnement à droite pour les mots AN et à gauche pour les mots NUMERIQUE.

. Si la zone émettrice est plus courte, sauf cadrage spécial, on fera un "remplissage" d'espaces à droite pour les mots AN et de zéros à gauche pour les mots NUMERIQUE.

Pour le verbe ARRET, on autorise que l'option FINAL.

Dans un premier compilateur le verbe LANGUAGE et les verbes de déclarations ne seront pas implantés.

4. 6. 2. <u>Première chaine codée et édition de la division TRAITEMENT</u>.

Chaque unité syntaxique sera représenté par un catène. Les mots clés (ou opérateurs) auront toujours un 1 en bit 24. Les huit bits suivants seront dévolus à la reconnaissance des mots que l'on classera en verbes, séparateurs etc...

Puis on affectera quatre bits à un poids de priorité qui permettra, au moment de la génération, de transformer la chaine codée en une chaine à notation post-fixée permettant des facilités de génération (cf IV. 6.).

Les opérandes auront toujours un 0 en bit 24. On reconnaitra leur genre par les bits suivants :

00 = nom de donnée

10 = libellés avec :
- 1000 = nombre entier
- 1001 = nombre réel
- 101 = libellé non-numérique.

01 = étiquette.

Les mots facultatifs du langage source seront éliminés

ainsi que les commentaires.

Chaque verbe sera affecté d'un numéro (croissant avec chaque occurrence d'un nouveau verbe) qui sera édité en fin de la ligne où l'on a rencontré le verbe.

Le problème de formalisation des séparateurs, des signes de ponctuation, des différentes parenthèses sera également résolu dans ce premier passage.

Grâce au verbe OUVRIR, on aura connaissance du genre des fichiers (ENTREE ou SORTIE) et ceci permettra de finir la codification de PCODFIC (cf IV. 4. 3. 2.) par le bit 23 mis en position 1 si le fichier est du type SORTIE.

En ce même passage seront formées les tables TRNND (cf IV 4. 4. 4) et les tables suivantes :

. Une table ETIQLEX correspondant à NDDLEX (cf IV. 4. 4. 2) pour les étiquettes dont le début est en colonne 8.

. Une table TRETIQ, correspondant à TRNND, pour les étiquettes suivant les verbes de branchement.

. Trois tables TBLE (libellés entiers), TBLR (libellés réels), TBLAN (libellés alphanumériques), qui emmagasineront les valeurs des différents libellés rencontrés.

Puis les tables NDDLEX et TRNND d'une part, ETIQLEX et TRETIQ d'autre part seront fusionnées en FUNND et FUETIQ respectivement (cf IV. 4. 4. 5.)

4. 6. 3. <u>Deuxième chaine codée et tables correspondantes</u>.

Le but de ce passage est d'améliorer la première chaine codée pour fournir une nouvelle chaine plus apte à être traitée de façon formelle. Pour expliquer ceci, prenons l'exemple suivant d'une instruction COBOL.

SOUSTRAIRE X DANS ANNEE, Y DE Z.

La première chaine a donné la suite de catènes (notation octale).

70100006	Correspondant à SOUSTRAIRE en supposant que ce verbe est le sixième rencontré depuis le début.
00000000	Correspondant au nom de donnée X DANS ANNEE.
00000000	" " " " " Y
51300000	" " mot-clé DE
00000000	" " nom de donnée Z
56204000	" " point

La deuxième chaine va remplacer cette instruction par :

SOUSTRAIRE - (X DANS ANNEE + Y) + Z RESULTAT Z.

c'est-à-dire par la suite de catènes.

70100006	Correspondant à soustraire et servant ici uniquement de repères pour les corrections éventuelles.
51260000	Correspondant au signe moins unitaire.
55000000	" à (
00000000	" " X DANS ANNEE
50760000	" " +
00000000	" " Y
55104000	" ")
50760000	" " +
00000000	" " Z
50050000	" " RESULTAT
00000000	" " Z
56204000	" " .

La supériorité de cette deuxième chaine sur la première est que tout opérande est toujours entouré de deux opérateurs ce qui rendra l'algorithme de transformation en notation post-fixée nettement plus simple.

Mais en même temps qu'il y a transformation de la chaine codée, il doit nécessairement y avoir transformation de la table FUNND.

Alors qu'elle était de la forme (par exemple) :

A006	pour X DANS ANNEE
B004	pour Y
D018	" Z

Elle deviendra :

A006
B004
D018
D018

Des transformations semblables seront réalisées pour les autres verbes arithmétiques et les verbes de transfert. Les abréviations des conditions composées seront également supprimées et les noms de condition remplacés par les expressions booléennes complètes qu'ils représentent.

4. 7. Conclusion sur la précodification.

On est parti du langage source tel que le programmeur l'écrit et on est arrivé à un certain nombre de tables qui représentent d'une façon concentrée, plus logique et ordonnée, les éléments du langage source.

Un grand nombre d'erreurs seront relevées pendant la précodification, pratiquement toutes les erreurs d'inattention, et un grand nombre d'erreurs syntaxiques.

°°°

V. CODIFICATION.

Partant des tables résultant de la précodification, cette partie du compilateur va :

. Réaliser un adressage des données ;

. Transformer définitivement la chaine codée pour en donner une nouvelle prête pour la génération ;

. Détecter les dernières erreurs.

5. 1. Codification de la division DONNEES.

Partant de la table TBPCOD, on veut aboutir à une codification regroupant tous les renseignements sur la description du mot ainsi que son adresse.

Cette codification sera formée, pour chaque mot, d'un groupement de sept à dix catènes comprenant toujours :

- 1 catène indiquant la longueur du groupement ;
- 1 catène EQLEX identifiant le groupement ;
- 1 catène PCOD ayant la structure de PCODO, amélioré si nécessaire par le passage interprétant les MODELES.
- 1 catène PCODAUX (mêmes remarques que pour PCODO) ;
- 1 catène LG donnant la longueur en caractères ;
- 1 catène ADDEBUT donnant l'adresse début du mot ;
- 1 catène ADCOMP donnant les renseignements complémentaires sur l'adresse c'est-à-dire :

. La longueur en catènes du mot, ou plutôt le nombre de catènes que chevauche le mot.

. La fraction de catène où débute le mot dans le premier catène.

. La fraction où finit le mot dans le dernier catène.

Au cas où le mot doit être indicié, les EQLEX des mots déterminant l'indice (c'est-à-dire les EQLEX des mots comportant la clause FIGURE) seront donnés.

Réalisation de l'adressage.

Pour avoir l'adresse complète, début et fin, d'un mot-groupe, il est évidemment nécessaire d'avoir déterminé précedemment les adresses début et fin des mots élémentaires qu'il contient. Le processus de réalisation de l'adressage va donc être similaire de celui qui a été réalisé pour les équivalents lexocographiques. A chaque n° de niveau on va associer cinq mémoires (de façon à former cinq stacks de cinquante mémoires) donnant respectivement :

. les adresses début des mots ;

. les adresses complémentaires début c'est-à-dire la fraction du premier catène appartenant au précédent.

. Les adresses complémentaires fin c'est-à-dire la fraction du dernier catène appartenant au mot traité.

. Les longueurs des mots, en catènes, arrondie au nombre

supérieur.

. Les longueurs de mots en caractères pour verification.

Lors de l'arrivée d'un mot élémentaire, on transformera son catène LG (longueur en caractères) en longueur en catènes et fractions de catènes s'il y a lieu - transformation réalisée grâce à la codification de la classe. (Dans le cas de mots PLACE ou NUMERIQUE INTERNE, cette transformation n'a pas à se faire, ou si elle se fait doit cadrer avec les impératifs du langage source). Puis on fera régresser, en modifiant s'il y a lieu les stacks d'adresse complémentaire, on codera tout ce qui est relatif au mot élémentaire, on modifiera l'ADDEBUT pour l'identificateur suivant et on fera une accumulation dans le stack des longueurs en catènes. La gestion de ce dernier stack est délicate car cette accumulation n'est pas simple. En effet prenons l'exemple suivant :

```
02   X;  AN
     03   Y; LG 2.
     03   Z; LG 6.
02   T.
```

La précodification a déterminé que les mots Y et Z étaient AN donc à 6 bits/caractères. Supposons que l'ADDEBUT soit à ce niveau de 66. Les stacks auront alors la forme suivante :

	ADDEBUT	ADCOMPD	ADCOMPF	LGCAT	LGCAR
(X)	66	0	0	0	0
(Y)	66	0	12	1	2
(Z)	66	12	0	2	6

Au moment du traitement de Z, ce qui correspond à Y sera effacé, mais il ne faut pas accumuler LGCAT(Z) + LGCAT(Y) = 3/2. Il faudra donc avoir une mémoire d'accumulation spéciale.

En fait la gestion de ce stack LGCAT, qui gère lui-même le stack ADDEBUT, sera rendu encore plus complexe par la présence des clauses FIGURE. En effet, si l'on avait eu :

03 Y LG 2 FIGURE 3 FOIS

les stacks auraient alors la forme suivante :

	ADDEBUT	ADCOMPD	ADCOMPF	LGCAT	LGCAR
(X)	66	0	0	0	0
(Y)	66	0	12	1	2
(Z)	66	12	0	2	6

et en remontant dans le stack

LGCAT(X) $\neq$ LGCAT(Y) x 3 + LGCAT(Z) = 3x1 +2 = 6 mais = 3.

On voit ici les difficultés spéciales à une machine à mots et les pourquoi de la restriction sur la cohabitation de mots à représentation internes différentes.

Quant au stack LGCAR, il ne sert de vérification que dans les cas où les longueurs des mots groupes ont été données.

Lorsqu'on rencontre un mot groupe, on traitera tous les mots groupes stackés précédemment de niveau inférieur à lui, en faisant les opérations nécessités par la clause FIGURE s'il y a lieu.

Il faut remarquer que si dans TBPCODO, les identificateurs étaient précodés dans l'ordre où ils étaient écrits, dans cette nouvelle table qu'on peut appeler TBDONCOD, ils sont dans l'ordre mot-élémentaire-mot-groupe. Ainsi dans notre exemple (cf IV. 4. 4.) les codifications seront dans l'ordre :

A004, A005, A003, A007, A008, A006, A009, A002, A011, A012, A010, A001, A014, A013, A000.

Les ADDEBUT sont des adresses relatives par rapport au début d'une zone, le nom de la zone se trouvant dans la table PCODFIC. La longueur de cette zone sera donnée ou vérifiée grâce au catène LGCAT de niveau 00.

5. 2. Fusion des tables TBCONDON et FUNND.

Lorsque l'on a obtenu les adresses de tous les identificateurs ainsi que leurs descriptions complètes on n'a intérêt à garder que celles qui serviront dans la suite, c'est-à-dire celles appartenant aux identificateurs dont les EQLEX sont dans FUNND. Dans le cas de variables indiciées, on remplace alors les EQLEX des noms à clause FIGURE, c'est-à-dire les un à trois derniers

catènes de la précodification par les ADCOMP respectifs des mots correspondants.

Afin d'optimiser cette fusion on réalisera une table analogue à TBALSUI donnant l'adresse de chaque début de codification de fichier dans TBDONCOD.

En vue de l'édition on réalisera une table d'équivalence entre les EQLEX et les adresses relatives.

5.3. Dernière chaine codée de la division TRAITEMENT. Erreurs syntaxiques.

Cette nouvelle transformation de la chaine codée a pour but d'introduire des instructions nouvelles telles que conversions de données d'une forme interne à une forme externe ou réciproquement cadrage de données et transfert dans des mémoires de manoeuvre, modification de l'ordre de certaines instructions afin d'obtenir un algorithme de transformation en chaine post-fixée plus simple, créations d'instructions plus près du code machine grâce à la codification de la division DONNEES, enfin détections d'erreurs syntaxiques qui n'avaient pu être trouvées jusque là, la codification de la division DONNEES n'apparaissant jamais en même temps que la chaine codée avant ce passage.

Explicitons en partie cette liste d'opérations sur l'exemple suivant :

MULTIPLIER X PAR Y RESULTAT Z ARRONDI DC I-1 SINON I-2.

I-1 et I-2 étant des instructions quelconques.

On suppose que par la codification des DONNEES, on a les renseignements suivants :

X est sous forme EXTERNE-1 avec	ADDEBUT	A + 72
	ADCOMPD	12
	ADCOMPF	16
	LGCAT	2

Y est sous la forme INTERNE-2 avec ADDEBUT = A + 78
ADCOMPD = 0 évident puisque
ADCOMPF = 0 sous forme
LGCAT = 2 INTERNE-2

Z est sous la forme EXTERNE-1 avec ADDEBUT = D + 4
ADCOMPD = 0
ADCOMPF = 16
LGCAT = 2
LGCAR = 10
VIRGULE GAUCHE = 2

Tous ces renseignements ayant été tirés de TBDONCOD.

La reconnaissance du code MULTIPLIER indique que tous les opérandes doivent être transformés sous forme flottante INTERNE-2.

Il faudra donc transformer X sous forme flottante en passant par l'intérmédiaire d'une mémoire M de longueur deux catènes qui cadrera ce nombre à droite sur un nombre entier de catènes ; puis transformer le contenu de cette mémoire M en une mémoire MFI (en réalité deux catènes) dont le contenu sera le mot sous forme flottante, en tenant compte de l'éventualité d'une clause virgule.

Le contenu du mot Y n'aura pas à être modifié et le programme object travaillera directement sur l'adresse délivrée par TBDONCOD.

Avant de réaliser l'opération RESULTAT Z, il faudra réaliser les opérations ARRONDI et DC. L'opération ARRONDI consiste à ajouter 5.10^{p-1} ou 5.10^{-v-1} (selon que l'on a l'option VIRGULE DROITE ou VIRGULE GAUCHE), p et v désignant les positions à droite et à gauche de la virgule, positions qui seront données par la codification. L'opération DC revient à tester s'il y a perte de poids fort dans le transfert occasionné par RESULTAT.

En supposant que $\rightarrow$ symbolise un transfert donnée-mémoire de manoeuvre et $\leftarrow$ un transfert mémoire de manoeuvre-donnée, la chaine sera transformée de la façon suivante :

MULTIPLIER X $\rightarrow$M $\rightarrow$ MF1$\times$Y ARR-V3 $\leftarrow$ M RESULTAT Z SI MFI $>$DC8 ALORS $\underline{T}$-1 SINON $\underline{T}$-2.

où ARR-V3 représente l'opération d'addition de 0.005 et DC8 est une mémoire contenant 10^8.

On verra sur le même exemple la transformation de cette chaine en notation post-fixée.

Une seule zone de transformation dans le sens -> est nécessaire. On notera la longueur maximum de cette zone telle qu'elle apparait dans une des instructions et au moment de la génération, on réservera une zone de cette longueur. On notera également le nombre maximum de bi-mémoires du genre MF1. A noter que dans les expressions arithmétiques (verbe CALCULER et clause SI), on ne peut à ce passage faire l'optimisation du nombre de ces mémoires, mais dans les autres verbes arithmétiques 2 bi-mémoires suffisent comme on le verra plus loin.

De même qu'on aura des mémoires flottantes, on aura des mémoires binaires, d'intérêt plus restreint et des mémoires booléennes lors de la génération.

La table TBCONDON sera complétée par l'insertion des différentes adresses et longueurs des mémoires de manoeuvre.

En même temps, des vérifications seront faites sur la rigueur des opérations. Ainsi dans notre exemple on aurait vérifié que tous les opérandes sont bien des mots élémentaires et que leur longueur ne dépasse pas l'équivalent de dix caractères décimaux.

o°o

6. GENERATION.

C'est dans cette dernière partie du compilateur que l'on doit procéder à la création d'instructions en langage objet. On part de la chaine codée, de la table TBDONCOD, et des différentes tables donnant les valeurs et les adresses des constantes, mémoires de manoeuvre et libellés.

La première partie de cette génération consistera à réserver et initialiser tous ce qui a rapport à ces dernières tables.

Le Gamma-60 ne possedant pas encore une gestion commerciale automatique simple et suffisamment proche de celle demandée par COBOL, il n'est pas possible de donner un aperçu de ce que serait la génération des entrées-sorties. De même le langage objet n'a jamais été vraiment défini, mais on peut supposer qu'il serait assez près du C7 ce qui permettrait de calquer de nombreuses générations d'instructions COBOL sur ce qui a été réalisé par ALGOL.

Comme pour ce dernier, on transformera la chaine codée en notation post-fixée suivant l'algorithme dont une version simplifiée est le compilogram BURROUGHS |20| que l'on peut résumer ainsi :

Les opérateurs ayant une priorité p définie lors de la codification et étant distingués des opérandes grâce à leur code (ici premier bit), on dispose de trois stacks dont :

- le premier contient la chaine codée ;
- le deuxième contient les opérateurs en attente ;
- le troisième emmagasine les opérandes à traiter ultérieurement.

Lorsqu'on rencontre un opérande on le descend dans le troisième stack et on fait progresser le premier.

Lorsque l'on rencontre un opérateur, on teste sa priorité par rapport à celle du dernier opérateur stacké dans le deuxième stack. Si elle est inférieure, on descend l'opérateur du deuxième au troisième stack et on recommence sur l'opérateur précédent du deuxième stack. Si elle est supérieure, on descend l'opérateur du 1er [illegible] vers le [illegible] r stack et on fait progresser le [illegible] 1er.

En réalité cet algorithme présente de nombreuses exceptions, tels que par exemple descente directe du premier stack vers le troisième dans le cas de certains opérateurs (ALORS, ARRONDI sauf après le verbe CALCULER...) traitement particulier du SI, éjections des verbes, traitements spéciaux pour les étiquettes suivies du code % etc...

Le tableau suivant donne les priorités d'un certain nombre d'opérateurs (les entrées-sorties ne sont pas inclues).

0	1	2	3	4	5	6	7
(	)	]	SI	ALORS	AUSSI	ET	OU
|	.	,	SINON	ALLER	JUSQUA		COMPTER
	;			PHRASE SUIVANTE	FOIS		REMPLACER
	:			ARRET FINAL	LIMITE		
				POUR PASSER A	AVEC		
				EXECUTER	DE(exécuter)		
				CONTINUER	PAR(exécuter)		
					A (exécuter)		
					SELON		

8	9	10	11
NON	=	RESULTAT	ARRONDI-n
TOUS	<	= (calculer)	
PREMIERS	>		
PREMIER	≠		
ARRETE A PREMIER	>0		
PAR (examiner)	<0		
	=0		
	ALPHABETIQUE		
	NUMERIQUE		

12	13	14	15
+	*	**	-->
-	/	Puissance	EN
⊕ (signe unitaire)	QUANTITE		<--
⊖ (" ")			⊏

Le signe ↓ est descendu dans le stack 2 après qu'un S est rencontré dans le stack 1.

Génération proprement dite.

Lorsqu'un opérateur a été descendu dans le stack 3, on nère l'opération attachée à cet opérateur et aux identificateurs qui le précèdent immédiatement. Dans notre exemple ceci donnerai

1 X --> M --> MF1 * Y ARR-3 <-- M RESULTAT Z SI MF1 > DC-8 ALORS T-1 SINON T-2

2 --> --> * <-- RESULTAT SI + ...

3 X M--> MF1 --> Y * ARR-3 M <-- Z MF1 DC-8 > ALORS T-1 SI RESULTAT 1-2 SINON.

Au point de vue génération on aurait les instructions suivantes (ou du moins les macro instructions).

X M --> résultat M
M MF1 --> " MF1 et la mémoire M redevient libre
MF1 Y * " MF1 " " " Y " "
ARR-3 " MF1
MF1 M -- " M " " " MF1 n'est pas libérée ; elle ne sera libérée que par un point ou un point virgule.
M Z MF1 DC-8 > résultat MBOOL et les mémoires MF1 et DC8 sont libérées
Supposons MBOOL faux on passe alors après SI
M Z RESULTAT Résultat Z et la mémoire M est libérée etc...

On voit que dans le cas d'un verbe AJOUTER ou SOUSTRAIRE, deux mémoires MF sont suffisantes.

Considérons l'exemple suivant :

AJOUTER X --> M --> MF1 + Y --> M --> MF2 + Z --> M --> MF2 RES : T

La notation post-fixée donne :

x M --> MF1 --> YM --> MF2 -- + ZM --> MF2 -- + T RES

et la génération

XM --> résultat M
M MF1 --> " MF1 et la mémoire M est libérée.
Y M --> " M
M MF2 --> " MF2 " " " M " "
MF1 MF2 + " MF1 " " " MF2 " "
Z M --> " M
M MF2 --> " MF2 " " " M " "

MF1 MF2 + Résultat MF1 et la mémoire MF2 est libérée

"

"

MF1 T RES " T

Libération de MF1.

On voit que à chaque addition la mémoire MF2 est libérée.

o°o

7. CONCLUSION SUR LA COMPILATION.

On n'a pas cherché à donner une méthode de compilation générale. L'exposé est très incomplet il resterait à éclaircir de nombreux points, ce qui ne pourra être fait qu'en mettant en pratique certains de ces passages. De plus un lien devra être trouvé entre les différents passages et il est possible que l'on puisse en contracter plusieurs en un seul, comme il est possible également que certains doivent être dupliqués comme par exemple pour les noms de condition.

Cependant on peut déjà se rendre compte de la complexité et de la longueur du problème dues ici principalement à la non-uniformisation des formes internes des mots à traiter, qui dépend non seulement du langage mais encore dans une grande part du calculateur.

o°o

CONCLUSION

La conclusion de cette étude devrait traiter de l'avenir de COBOL. Encore faut-il distinguer, à l'heure actuelle, les conclusions sur le langage source et celles sur le système complet.

1. LANGAGE SOURCE.

Nous avons vu que COBOL présentait un nombre très limité d'ambiguïtés, et que, pour la division TRAITEMENT tout au moins, on pouvait en donner une définition suivant la métalangue utilisée pour décrire ALGOL. En se limitant au point de vue définition ri-

goureuse du langage, on peut dire que COBOL (à quelques exceptions près) ne présente pas de défauts.

Puisque nous avons un langage rigoureux, on pourrait penser qu'il va servir de moyens de communication entre utilisateurs préoccupés par les mêmes genres de problèmes. En fait, on ne peut comparer COBOL à un langage tel qu'ALGOL pour des échanges de programmes car, si des algorithmes numériques peuvent se communiquer dans toute leur généralité, il est difficile d'invoquer une généralité, pouvant ensuite se matérialiser, lorsqu'il s'agit de problèmes commerciaux. De plus, nous avons vu les limites qu'il fallait apporter à la notion de compatibilité et les dangers qui peuvent résulter d'une transplantation sans changements quand on passe d'une machine à une autre.

Quelles sont donc les perspectives d'utilisation de COBOL ? Tout d'abord, si un langage commercial doit être adopté comme langage commun, on peut dire que COBOL présente de nombreux avantages sur les autres langages commerciaux, avantages dont les principaux sont (cf ch. II) : clarté, étendue des possibilités offertes au programmeur, compatibilité maximum, définition rigoureuse et surtout existence prévue, non pour une seule machine mais pour n'importe quelle machine.

Imposé en Amérique par le Département de la Défense, qui refuse d'acheter toute machine n'ayant pas un traducteur COBOL, le le langage semble prêt à une bonne carrière, malgré ses nombreux détracteurs, principalement les Compagnies ayant écrits d'autres traducteurs tels que FACT, tout aussi orientés, mais moins communs, et de ce fait plus performants. D'autre part, l'échelle des utilisateurs américains, disposant de plusieurs machines dans une installation, nécessite un langage de communication, tout au moins au niveau de l'analyse et COBOL peut parfaitement jouer ce rôle.

Quant au COBOL - français (et plus généralement de langue non-anglaise) son avenir est plus incertain. Après une assez longue période d'ignorance, les constructeurs français se sont décidés à normaliser (par l'ECMA) une traduction des clichés originaux. Mais

pour l'instant, seuls les constructeurs connaissent COBOL. Assez peu fiers de leurs traducteurs, ou n'en ayant pas écrits, ils n'avaient fait jusqu'à il y a quelques mois, aucune publicité sur l'existence de COBOL et de ce fait les utilisateurs n'avaient aucune idée des services que pourraient leur rendre COBOL. D'autre part, les entreprises françaises ne possédant en général qu'une seule machine, la compatibilité les intéresse peu et seules les performances -toujours en progression - d'autocodeurs assez simples leur semblent suffisantes. C'est pourquoi, on ne pourra lancer véritablement COBOL en France, que le jour où les traducteurs seront suffisamment adaptés aux exigences des problèmes de gestion.

2. LES TRADUCTEURS.

Il existe des traducteurs COBOL, mais aucun des réalisateurs de ces traducteurs n'a exposé la méthode de compilation. Les quelques renseignements que l'on a pu trouver sur des traducteurs existant donnent des temps de compilation et des nombres de passages exhorbitants. D'autre part, le programme objet est toujours loin d'être performant.

Or si l'on peut supposer qu'un algorithme scientifique ne sera passé en machine qu'un nombre de fois excessivement restreint, et donc que son temps d'exécution n'est pas le facteur primordial, il n'en est plus de même lorsqu'il s'agit d'un problème de paye ou de gestion de stocks qui doit être repassé périodidiquement. On voit donc l'importance des temps d'exécution et de l'optimisation du programme généré dans les cas de langages symboliques commerciaux.

Partant de là, que peut-on en conclure sur la traduction de COBOL en général, et de sa compilation sur le Gamma-60 en particulier.

Tout d'abord, et nous avons vu pourquoi au moment de l'adressage des données et de la création d'instructions supplémentaires (cf ch. IV), la traduction sur une machine à mots sera

plus compliquée que celle sur une machine à caractères. La première impose des conventions nouvelles sur la place des mots-COBOL dans le mot-machine, donc des adjonctions au rapport original (prévues d'ailleurs par celui-ci), mais également des restrictions si l'on veut avoir un programme objet performant. Il faudra créer des instructions supplémentaires de splittage et cadrage de mots, et une connaissance approfondie de la machine sera nécessaire pour que le programmeur donne à chaque mot, un USAGE approprié au traitement qu'il lui fera subir. Ceci imposera d'ailleurs une écriture plus lourde de la division DONNEES, sans pour autant donner des difficultés de lecture supplémentaires.

Le Gamma-60 lui, présente le maximum d'inconvénients pour la description des données, puisque les caractères alphanumériques et alphabétiques n'occupent pas le même nombre de bits dans le mot-machine et que toute application numérique se fait en semi-logarithmique. On va donc avoir des USAGES nombreux et dans le cas d'un programmeur ne connaissant pas suffisamment la machine, un nombre de conversions internes énorme qui fera perdre toute efficacité au programme généré.

Pour que la compilation d'un langage tel que COBOL soit rentable, il faut pouvoir l'inclure dans un système complet dans le cas de très grosses machines qu'il puisse se rattacher à un système d'entrée-sortie convenable comme l'I.O.C.S. par exemple. Malheureusement, il n'existe encore rien de tel pour le Gamma-60, ce qui augmente encore la difficulté de traduction.

Enfin, il reste à discuter le nombre de passages que prendra la traduction. Théoriquement, avec une mémoire de capacité suffisante, il suffirait d'un seul passage puisque le problème est statique. Mais la lourdeur, longueur et verbosité du langage source font qu'il est impossible de concentrer et utiliser, ou stocker, simultanément toutes les informations nécessaires, aucune machine à l'heure actuelle n'ayant une mémoire centrale suffisamment grande. Il sera donc nécessaire d'utiliser des mémoires auxiliaires (bandes ou tambours) et le nombre de passages en mémoire centrale dépendra

de l'organisation choisie et bien entendu de la capacité de la mémoire centrale. De toute façon, il semble que pour l'instant COBOL ne puisse être utilisé que sur des machines de moyenne ou grande capacité si l'on ne veut pas faire trop de restrictions au langage source, et pour cela le Gamma-60 semble approprié.

o°o

Au moment où l'on essaye de normaliser COBOL, ces conclusions paraissent pessimistes. Si la normalisation semble bien prématurée, vu les addendas apportés au langage source à l'heure actuelle et les modifications qui seront nécessairement faites après une certaine expérience de l'utilisation de COBOL, il n'en reste pas moins que le meilleur argument en faveur de COBOL est son existence même. Elle empêche en effet la création de langages spécifiques orientés vers une seule machine, et au lieu d'avoir des autocodes se rapprochant plus ou moins d'un véritable langage de programmation, les constructeurs pourront fabriquer des COBOL, même restreints, ce qui évitera une prolifération bien inutile.

Cependant, de nombreuses améliorations sont encore à faire, ou en train d'être faites, et l'on ne pourra véritablement juger le langage que lorsqu'il aura été utilisé à une échelle supérieure de ce qui est fait actuellement. C'est pourquoi, il faudrait intensifier la publicité sur le langage, que les constructeurs améliorent, ou fassent, des traducteurs, et que des expériences nombreuses soient tentées avant de porter un jugement définitif, à savoir si COBOL est bien le modèle du langage orienté vers les applications commerciales.

o°o

APPENDICE 1

. Clichés de la division EQUIPEMENT.

1. 1. Section CONFIGURATION.

1.1.1. § COMPILATION

ption 1 COMPILATION. COPIER bibliothèque.

ption 2 COMPILATION. calculateur [AVEC SUPERVISEUR]

{ ; MEMOIRE entier-1 { MOTS / CARACTERES / MODULES }
ADRESSE DEPUIS entier-2 JUSQU'A entier [, DEPUIS ...] }

[; [entier -4] élément-1] [; [entier-5] élément-2]

1. 1. 2. § EXECUTION.

ption 1 EXECUTION. COPIER bibliothèque.

ption 2 EXECUTION. calculateur [AVEC SUPERVISEUR]

[; MEMOIRE cf option 2 § 1.1.1.]

[; [entier-4] cf 1.1.1.]

[; SEGMENTER DEPUIS entier-8] ; [METTRE PROGRAMME SUR élément d'entrée.]

1. 1. 3. § NOMS-SPECIAUX.

ption 1 NOMS-SPECIAUX. COPIER bibliothèque.

ption 2 NOMS-SPECIAUX. élément-1 EST DESIGNE PAR nom mnémonique-1
[, élément-2 EST ...].

ption 3 NOMS-SPECIAUX.

élément-1 { [EST DESIGNE PAR nom mnémonique-1]
[EN FONCTION EST DESIGNE PAR condition 1]
[HORS FONCTION EST DESIGNE PAR condition 2] }

[élément-2 ...].

1. 2. Section ENTREE-SORTIE.

1. 2. 1. § GESTION-FICHIERS.

ion 1 GESTION-FICHIERS. COPIER bibliothèque.

ion 2 GESTION-FICHIERS. PRENDRE [EVENTUEL] fichier-1 [COPIE DE fichier-2]

METTRE SUR [entier-1] élément-1 [élément-2]

[PLUSIEURS BOBINES] [NE RESERVER {entier-2 / AUCUNE} [{ZONE SUPPLEMENTAIRE / ZONES SUPPLEMENTAIRES}]

[PRIORITE priorité] . [PRENDRE...] .

1. 2. 2. § GESTION-E-S.

tion 1 GESTION-E-S. COPIER bibliothèque.

tion 2 GESTION-E-S. [EMPLOYER technique POUR fichier-1] ; [EMPLOYER ..

[REPRISE [UTILISANT {fichier-2 / élément} CHAQUE {FIN DE BOBINE / entier-1 ARTICLES} DE fichier-

[MEME ZONE [ARTICLE {UTILISE / UTILISEE} PAR fichier-4, fichier 5 , [fichier-...

[; BANDE PLUSIEURS FICHIERS CONTIENT fichier-7 [POSITION entier-

[, fichier-8 [POSITION entier-4]...]].

Clichés de la division DONNEES.

2. 1. Description de fichier.

tion 1 DF fichier COPIER nom dans bibliothèque.

tion 2 DF fichier {REPERE / REPERES} {STANDARD / OMIS / donnée-1 / nom dans biblio-1 DANS BIBLIOTHEQUE}

[{donnée-2 / nom dans biblio-2 DANS BIBLIOTHEQUE}]

; {NOM / NOMS} DES ARTICLES donnée-3 [, donnée-4 ...]

[; BLOC CONTIENT [entier-1 A] entier-2 {ARTICLES / CARACTERES}]

[; FICHIER CONTIENT ENVIRON entier-3 ARTICLES]

[; ARTICLE CONTIENT [entier-4 A] entier-5 CARACTERES]

[; ENREGISTRE EN MODE mode] ; [TRIE SUR donnée-5 , [donnée-6] ..

[; VALEUR DE donnée-7 { donnée-8 / libellé-1 } [CUMULE] , } [donnée-9] ...] .

2. 2. Description d'article courant.

tion 1 01 donnée-1; COPIER donnée-2 [DEPUIS BIBLIOTHEQUE] .

tion 2 01 donnée-2; [LONGUEUR ...] [CLASSE ...] .

2. 3. Description de mot groupe.

tion 1 NN donnée-1; [REDEFINIT donnée-2] ; COPIER donnée-3 [DEPUIS BIBLIOTHEQUE.]

tion 2 NN donnée-2; [REDEFINIT donnée-3] ; [LONGUEUR ...] ; [CLASSE ...
[; FIGURE ...] ; [USAGE ...] .

tion 3 66 donnée-1; RECOUVRE donnée-2 [JUSQU'A donnée-3] .

2. 4. Description de mot élémentaire.

tion 1 cf option 1 mot groupe.

tion 2 NN { donnée-1 / INDEFINI } [REDEFINIT donnée-2] { LONGUEUR / LG } entier-1 [A entier 2 { CARACTERES / CHIFFRES }]
[SELON donnée-3]

; CLASSE { ALPHANUMERIQUE / AN / ALPHABETIQUE / NUMERIQUE } ; [USAGE { EXTERNE / EXTERNE-n / INTERNE / INTERNE-n }]

[; FIGURE entier-3 [A entier-4] FOIS [SELON { condition / donnée-4 }]]

[; { ALGEBRIQUE / SIGNE EN donnée 5 }] ; [CADRE { SUR / VERS } LA { GAUCHE / DROITE }]

[; VIRGULE { SUR / VERS } LA { GAUCHE / DROITE } entier-5 { POSITIONS / BITS }]

[; PLACE { SUR / VERS } LA { GAUCHE / DROITE }] ; [ETENDUE DEPUIS libellé-1 [JUSQU'A libellé-2]]

[; { ZEROS SUPPRIMES / PROTEGER / SYMBOLE MONETAIRE FLOTTANT } [EN GARDANT entier-6 POSITIONS]]

[; BLANC { QUAND / LORSQUE } ZERO ; [MODELE (Toute combinaison permise de caractères ou symboles décrite en VI.33 du rapport COBOL.) [SELON donnée

2. 5. Description des noms de condition.

ion 1 88 donnée; VALEUR libellé.

ion 2 88 donnée; { VALEUR / VALEURS } [DEPUIS] libellé 1 [JUSQU'A Libellé-2 / DEPUIS ...]

Division TRAITEMENT.

3. 1. Verbes arithmétiques.

{ ADDITIONNER / AJOUTER } { libellé-1 / donnée-1 } [{ libellé-2 / donnée-2 } ...] [{ A / RESULTAT } donnée-n]

[ARRONDI] [{ LORSQUE / QUAND } { DEPASSEMENT / DC } toute instruction impérative]

SOUSTRAIRE { libellé-1 / donnée-1 } [{ libellé-2 / donnée-2 } ...] DE { libellé-n / donnée-n }

[RESULTAT donnée-m] [ARRONDI] ; { DEPASSEMENT / DC } ....

MULTIPLIER { libellé-1 / donnée-1 } PAR { libellé-2 / donnée-2 } [RESULTAT donnée-3] [ARRONDI

[; { DEPASSEMENT / DC } ...]

DIVISER-PAR { libellé-1 / donnée-1 } QUANTITE { libellé-2 / donnée-2 } [RESULTAT donnée-3] [ARRONDI

[; { DEPASSEMENT / DC } ...]

DIVISER { libellé-1 / donnée-1 } PAR { libellé-2 / donnée-2 } [RESULTAT donnée-3] [ARRONDI]

[{ ; DEPASSEMENT / DC } ...]

CALCULER donnée-1 [ARRONDI] { DE / = / EGALE } { donnée-2 / expression } ; [{ DEPASSEMENT / DC } ...]

2. Verbes d'entrée-sortie.

OUVRIR POUR { [ENTREE fichier-1 [INVERSE] , [fichier-2] ...] / [SORTIE fichier-3 [, fichier-4 ...]] }

FERMER POUR fichier-1 [BOBINE COURANTE { SANS REBOBINAGE / PUIS VERROUILLER }] [POUR ...]

LIRE fichier [PUIS TRANSFERT VERS donnée { LORSQUE / QUAND } { FIN DE FICHIER / FF / instr. impér. }]

ECRIRE article [DEPUIS donnée-1 { AVANT / APRES } SAUT { donnée-2 / entier } { INTERLIGNES / INTERLIGNES / nom mnémomique }]

RECEVOIR donnée [DE nom mnémomique]

INDIQUER { donnée-1 / libellé-1 } , [{ donnée-2 / libellé-2 } ...] [SUR nom mnémomique]

3. Verbe de branchement.

1 ALLER A étiquette

2 ALLER A étiquette-1, étiquette-2 [, étiquette-3 ...] SELON donnée

1 EXECUTER étiquette-1 [JUSQU'A étiquette-2]

2 EXECUTER étiquette-1 [JUSQU'A étiquette-2] { donnée-1 / entier } FOIS

3 EXECUTER étiquette-1 [JUSQU'A étiquette-2] {ARRETER / LIMITE} [LORSQUE / QUAND] condition

4 EXECUTER étiquette-1 [JUSQU'A étiquette-2] AVEC donnée-1 VARIANT DE {donnée-2 / libellé-1} PAR {donnée-3 / libellé-2} {ARRETER / LIMITE} [LORSQUE / QUAND] condition

5 EXECUTER étiquette-1 [JUSQU'A étiquette-2] AVEC indice-1 VARIANT DE {donnée-1 / libellé-1} PAR {donnée-2 / libellé-2} {ARRETER / LIMITE} [LORSQUE / QUAND] condition-1

[APRES indice-2 VARIANT DE { } PAR { } {ARRETER / LIMITE} [LORSQUE / QUAND] condition-2

[APRES indice-3 VARIANT DE { } PAR { } {ARRETER / LIMITE} [LORSQUE / QUAND] condition-3]

CHANGER étiquette-1 POUR PASSER A étiquette-2 , [étiquette-3 POUR ...]

4. Verbes de transfert.

TRANSFERER {[LORSQUE / QUAND] [CONCORDANCE] donnée-1 / libellé-} EN donnée-2 , [donnée-3 ...]

EXAMINER donnée POUR {
{COMPTER / COMPTACE} {ARRETE A PREMIER / TOUS / PREMIERS} libellé-1
[PUIS {REMPLACER / REMPLACEMENT} PAR libellé-2]
{REMPLACER / REMPLACEMENT} [TOUS / PREMIERS / ARRETE A PREMIER] libellé-3 PAR libellé-4
}

5. Verbe de gestion du compilateur.

LANGAGE langage [programme]

CONTINUER

NOTE

INCLURE étiquette [, PUIS REMPLACER { mot programme 1 / donnée-1 } PAR { mot-1 / donnée-2 } [, { mot programme-2 / donnée-3 } ...]]

6. Verbe d'arrêt :

ARRET { libellé / FINAL }

7. Verbes de déclaration :

1 UTILISER APRES GESTION ERREUR POUR { fichier / ENTREE / SORTIE }

2 UTILISER { AVANT / APRES } GESTION [{ DEBUT / FIN } { BOBINE / FICHIER }] STANDARD POUR { fichier / ENTREE / SORTIE }

DEFINIR verbe FORME verbe

o°o

APPENDICE 2

Compilateurs COBOL-61

onstructeurs	Modèle	Mémoires	Bandes	Date de mise en service présumée
NDIX	G-20	8.192	5	Déc. 62
RROUGHS	B-220	5 000	3	Déc. 62
	B-5000	4 096	3 (+1 tambour)	Mars 63
NTROL DATA	1604	32 768	8	Janv. 63
	3600	32 768	8	Mai 63
NERAL ELECTRIC	GE-225	8 192	2	En service
B. M.	1401	12 000 K	4	En service
	1410	20 000 K	4	En service
	7040/44	16 384	5	Juillet 63
	709/90	32 768	5	Déc. 62
	7070/74	10 000	7	En service
NEYWELL	H-400	2 048	4	Janv. 63
	H-800	4 096	4	Janv. 63
C R	304	2 600	6	En service
	315	20 600	5	En service
ILCO	2000	8 192	8	En service
C A	301	20 000K	6	Janv 63
IVAC	1105	12 000	10	En service
	1107	16 384	6	Déc. 62
LVANIA	9400	16 000	6	Janv. 63

°°°

BIBLIOGRAPHIE

Report to Conference on Data Systems Language In cluding Revised Specifications for a Common Business Oriented Language for Programming Digital Compaters.
Department of Defense U. S. A. 1961

E C M A - T C 6 - Traduction française de COBOL. Oct. 1962.

UNIVAC - Flowmatic Programming Systems - Remington Rand Univac Publication - U 1518 - 1958

I. B. M. The Commercial Translator - General Information Manuel - I. B. M. Publication F 28-8013 - 1959

R. F. CLIPPINGER. FACT : A Business Compiler - Description and Comparison with COBOL and Commercial Translator.
Annual Review in Automatic Programming - Vol. 2 1961.

T. G. H. BRAUNHOLTZ, A. G. FRASER, P. M. HUNT - NEBULA , a programming language for data progressing.
The Computer Journal - Oct. 1961 - p. 197 - 211

A.LIPPITT - COBOL and Compatibility
Comm. of the A. C. M. Mai 1962 p. 254 - 255.

I. C. T. Rapidwrite Programming Manual - 1961.

E. L. WILLEY et al. Some Commercial Autocodes - A comparative Study -
Academic Press 1961

P. NAUR et al. Report on the Algorithmic Language ALGOL 60.
Comm of the A. C. M. Mars 1960 p. 299 - 314

M. F. GENUYS et al. Rapport sur le langage algorithmique ALGOL 60.
Chiffres - 3 - 1960 - p. 1 - 44.

GROUPE ALGOL GRENOBLE - Manuel de Programmation ALGOL - Oct. 1962.

J. E. SAMMET - A method of Combining ALGOL and COBOL.
Sylvania Electronics Systems - Mai 1961.

4| J. E. SAMMET. A definition of the COBOL-61 Procedure Division using ALGOL-60 metalinguistics.
Sylvania Electronics Systems - Sept. 1961.

5| R. BERMAN, J. SHARP, L. STRURGES - Syntactical Charts of COBOL - 61.
Comm. of the A. C. M. Mai 1962 p. 200

6| I. B. M. Notice explicative - 1401 - CB - 071.

7| R. C. A. COBOL compiler write - UPS. Avril 1962.

8| T. A. DOLOTTA. Les langages symboliques et leur édition.
Chiffres 3-1962. p. 149-174.

9| M. E. COWWAY, J. SPERONT, Arithmetizing Declarations. An application to COBOL.
Com of the A. M. A. Janvier 1963.

0| BURROUGHS - The compilogram from Burroughs. A game to Counter Compilerits.
Datamation - Février 1962 - p. 32-33.

°°°

TABLE DES MATIERES.

Introduction.

Chapitre I. Présentation rapide de COBOL. 1

1. Structure générale de COBOL. 3
2. Division Données. 9
3. Les divisions Identification et Equipement. 10
4. La division Traitement. 19
5. Caractéristiques particulières.

Chapitre II. Etude critique de COBOL. 21

1. Introduction 21
2. La compatibilité dans COBOL 24
3. COBOL outil d'analyse et de programmation 26
4. Longueurs, redondances, imprécisions 32
5. Comparaison avec les autocodes commerciaux 37
6. Avantages et inconvénients de COBOL, langage commercial.

Chapitre III. Comparaison d'ALGOL et de COBOL. 39

1. Introduction. 39
2. Comparaison des métalangues ALGOL et COBOL. 42
3. Définition de la division traitement de COBOL suivant la métalangue ALGOL. 52
4. Extension possible aux autres divisions de COBOL. 53
5. Comparaison entre ALGOL et la division traitement de COBOL. 60
6. Conclusion.

Chapitre IV. Quelques données pratiques sur la compilation de COBOL sur une machine à mots. 62

1. Définition du langage source à compiler. 64
2. Organisation générale de la compilation. 65
3. Extensions et premières vérifications sur le langage source. 66
4. Formalisation et précodification. 85
5. Codification. 91
6. Génération. 96

Conclusion. 100

Appendice I. 107

Appendice II.