THÈSE DE SPÉCIALITÉ
mathématiques appliquées
INFORMATIQUE

UNIVERSITÉ DE NANCY I

Patrick BONNET

«Assistance informatique à la stratégie de synthèse en chimie des organophosphorés»

soutenue le 27 février 1978 devant la commision d'examen:

M. C. PAIR — Président

M. J.C. DERNIAME, M. G. KAUFMANN — Examinateurs

THÈSE DE SPÉCIALITÉ
mathématiques appliquées
INFORMATIQUE

Patrick BONNET

—

UNIVERSITÉ DE NANCY I

—

«Assistance informatique à la stratégie de synthèse en chimie des organophosphorés»

soutenue le 27 février 1978 devant la commision d'examen:

M. C. PAIR	Président
M. J.C. DERNIAME	Examinateurs
M. G. KAUFMANN	

à mes parents,
à qui je dois tant

Ce travail a été réalisé à l'Université de Nancy I, sous la direction de Monsieur le Professeur J.C. DERNIAME. Le sujet de l'étude, ainsi que sa réalisation pratique, s'inscrit dans le cadre des recherches menées par le "Laboratoire de Modèles Informatiques appliqués à la Synthèse" de l'Université Louis Pasteur de Strasbourg, sous la direction de Monsieur G. KAUFMANN, Maître de Recherche au CNRS.

Qu'il me soit permis d'exprimer ici toute la gratitude que je dois à Monsieur DERNIAME pour le temps qu'il a bien voulu me consacrer, pour son aide et pour les conseils précieux qu'il m'a donnés tout au long de mon travail.

Je voudrais aussi témoigner la profonde reconnaissance que je porte à Monsieur KAUFMANN, pour les conditions exceptionnelles dans lesquelles il m'a permis de réaliser mon travail, ainsi que pour son aide et sa disponibilité.

Je voudrais dire également à quel point il m'a été agréable de travailler au sein de son équipe ; je remercie tout particulièrement Mademoiselle M.H. ZIMMER et Messieurs F. CHOPLIN , C. LAURENÇO et R. MARC pour leur précieuse collaboration et pour la compréhension qu'ils ont témoignée face à mon faible niveau dans leur discipline.

Je suis reconnaissant à Monsieur le Professeur C. PAIR, de l'Université de Nancy I, qui a accepté de juger ce travail et de présider le jury.

Je tiens enfin à remercier vivement tous ceux qui ont participé d'une manière ou d'une autre à la réalisation pratique de ce travail, en particulier Mademoiselle M. LAUTH à qui je dois la dactylographie remarquable de cet ouvrage.

Je terminerai par une pensée particulière pour tous ceux qui ont participé à ma formation.

TABLE DES MATIERES

AVERTISSEMENT

Le système informatique dont le développement est présenté ici étant destiné à des utilisateurs chimistes, il nous a paru important de donner une maximum d'exemples intéressants destinés au lecteur chimiste. Soulignons toutefois que c'est l'aspect informatique qui est présenté ; en conséquence, même ces exemples sont décrits de telle sorte qu'ils puissent rester compréhensibles pour des personnes qui n'auraient que de faibles notions en chimie. La bibliographie donne les références à consulter pour avoir une vision de ce système sous son aspect chimique.

* * *

INTRODUCTION

L'établissement des chemins de synthèse conduisant à une molécule donnée est une opération complexe faisant appel simultanément à la logique, à l'intuition et aux connaissances du chimiste.

Ainsi, dans le domaine des molécules d'origine naturelle (vitamines, antibiotiques, prostaglandines, hormones...), la création de schémas synthétiques se révèle particulièrement ardue tant par le grand nombre de chemins de synthèse envisageables que par les contraintes importantes liées à l'environnement économique : coût des matières premières, nombre d'étapes réduit pour un processus industriel, etc...

Afin d'accélérer la planification des chemins de synthèse de molécules organiques, des systèmes informatiques ont été proposés récemment pour assister les chimistes dans cette opération. Les concepts fondamentaux mis en oeuvre en chimie étant graphiques par excellence, on s'est orienté vers l'emploi de consoles graphiques, sur lesquelles il est facile de représenter des molécules ou des sous-structures par leur dessin tout à fait classique, qui traduit tant leur topologie que des paramètres tels que le type des atomes, la multiplicité des liaisons, etc...

Ainsi, en chimie organique, les programmes OCSS , puis LHASA (Corey) (1), SECS (Wipke) (2) (voir aussi (3) à (8)) simulent d'une façon plus ou moins efficace la démarche humaine face à un problème de synthèse ; d'autres systèmes tels DARC (9) peuvent apporter une aide importante en matière de documentation automatique ou de conception assistée.

Dans le domaine de la synthèse assistée, le laboratoire de Modèles Informatiques appliqués à la Synthèse (ERA 671) travaille à la réalisation du Programme d'Assistance à la Synthèse en Chimie Organophosphorée (PASCOP) (12 et 26).

PASCOP, comme les systèmes antérieurs, utilise le principe de l'analyse rétrosynthétique, c'est à dire que les précurseurs de la molécule à synthétiser (la cible) se déduisent de celle-ci en lui appliquant des opérations topologiques appelées transformations et qui représentent des réactions chimiques prises dans le sens inverse de leur utilisation synthétique.

Ainsi, les précurseurs d'un alcool de forme RR'R"C - OH peuvent être déterminés en appliquant à cette cible la transformation

$$RR'R"C - OH \Longrightarrow RMgX + R'R"C{=}O$$

qui correspond à la réaction chimique

$$RMgX + R'R"C{=}O \longrightarrow RR'R"C{-}OH$$

En exploitant un fichier de transformations, le système construit par itérations successives un graphe appelé arbre de synthèse où la racine correspond à la cible, les divers noeuds aux précurseurs et les arcs aux transformations retenues.

Si un tel système permet de construire des chemins de synthèse originaux, il ne permet pas à l'utilisateur de diriger a priori la création de l'arbre de synthèse par la mise en oeuvre par exemple de stratégies appropriées.

Le présent travail décrit le développement d'un module de stratégie destiné à accélérer la recherche par PASCOP de chemins de synthèse optimaux. Il s'agira tout d'abord de munir ce système d'une base solide sur laquelle s'appuyera un module capable de diriger la construction de l'arbre de synthèse en prenant en compte d'une façon interactive des directives formulées par le chimiste, ou en exploitant des automates spécialisés dans l'application d'une transformation jugée prioritaire sur ce plan synthétique. A partir de là, on pourra lui adjoindre des modules de guidage automatique en fonction de notions telles que coût de produits, rendements, etc...

A côté de cela, il a fallu définir un outil permettant la communication entre ce système spécialisé et son interlocuteur humain et non informaticien.

Ce travail sera présenté en quatre chapitres :

- le premier décrit précisément les problèmes vus sous l'angle chimique, donc utilisateur. On y verra également le fonctionnement actuel du système PASCOP
- le deuxième présente les choix faits pour donner au système ses possibilités d'actions en stratégie
- dans le troisième, on étudiera les moyens mis en oeuvre pour résoudre les problèmes de communication homme-système
- quant au quatrième, il présentera le langage STRATOS défini pour cette communication, tel que l'utilisateur sera amené à l'aborder.

CHAPITRE I

EXPOSE PRECIS DU PROBLEME

11 - PRESENTATION DE CE DONT ON DISPOSE AU DEPART : LE SYSTEME PASCOP

Le but du système PASCOP est de permettre la recherche de chemins de synthèse de molécules organo-phosphorées. Il peut fournir en outre à l'utilisateur des références bibliographiques concernant les réactions employées dans les chemins de synthèse proposés, afin de faciliter leur réalisation pratique.

Ce système est dérivé du programme SECS (Simulation and Evaluation of Chemical Synthesis), version 2, de 1975, réalisé à Princeton pour la synthèse en chimie organique.

PASCOP fonctionne sur un ordinateur Univac 1110 connecté à un PDP 11/10 qui assure le pilotage d'un écran GT 40 avec crayon lumineux, et d'un télétype (10,11).Le système occupe environ 93 K mots de 36 bits, réduits à 54 K effectifs par un découpage en phases.

111 - Description sommaire de l'utilisation de PASCOP

* 1ère PHASE : Introduction de la cible à atteindre

Elle se passe en deux parties. Tout d'abord, dessin de la cible sur l'écran, à l'aide du crayon lumineux, puis "perception" de cette cible par le système.

En fait le dessin de la structure de la cible est régi par des règles précises, ce qui permet de faire appel pour sa réalisation à des fonctions élémentaires sans passer par un programme de reconnaissance de formes.

Ces fonctions élémentaires sont appelées par les commandes :

- positionnement d'un nouvel atome, supposé lié à un atome précédent déjà positionné
- tracé d'une liaison double ou triple
- positionnement d'un atome sans liaison avec le précédent
- définition du type d'un atome (oxygène, azote, phosphore, halogène,...), les atomes définis sans type étant considérés comme des atomes de carbone
- dessin d'un cycle à 5 ou 6 membres, en en désignant simplement le centre
- définition de liaisons en avant ou en arrière du plan de l'écran
- effacement partiel ou complet du dessin
- déplacement d'atomes sur l'écran.

L'utilisateur choisit la commande désirée en en pointant le nom à l'aide du crayon lumineux ; on sort d'une commande en pointant le crayon hors du cadre de dessin, à gauche.(Exemple planche A).

Au cours du tracé de la molécule, le système en construit une représentation interne sous forme d'une table de connexions (voir § 1131), et d'une table des coordonnées-écran des atomes dans un repère xOy dont l'origine O est située au coin inférieur gauche de l'écran.

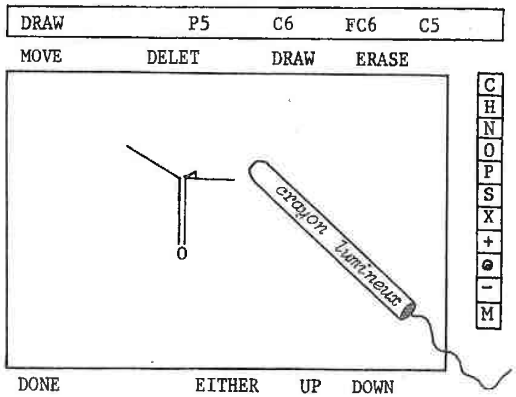
Inversement, un sous-programme permet de recréer sur l'écran la structure de la cible à partir de ses tables.

L'utilisateur peut sauvegarder les tables dans un fichier en cas d'arrêt momentanné du travail.

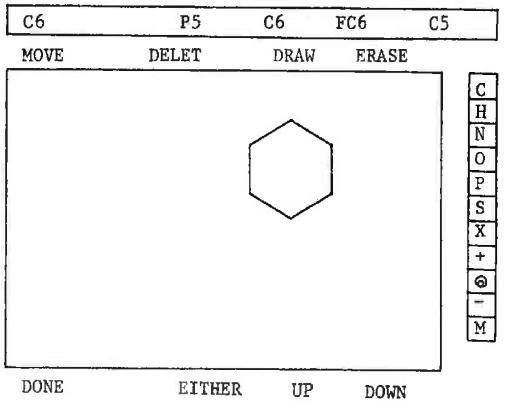
La saisie de la cible est achevée par la commande DONE, qui permet de passer au traitement synthétique de la cible. Cette opération débute en soumettant la cible à une perception au cours de laquelle l'information topologique est transformée en une information à caractère chimique. On détermine notamment les cycles de base ainsi que leur caractère aromatique ou anti-aromatique, et les divers groupes fonctionnels présents dans la cible.

* 2ème PHASE : Recherche et évaluation de précurseurs

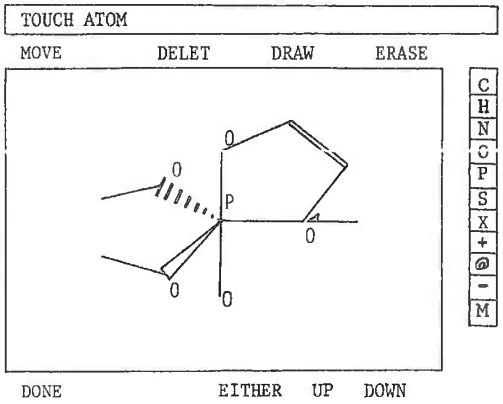
Rappelons qu'une structure est un précurseur de la cible proposée s'il existe une transformation permettant de passer de la cible à



- fonction sélectionnée = DRAW
- création d'un nouvel atome à l'endroit pointé par le crayon lumineux,avec tracé de la liaison avec le précédent, encore désigné par le sytème par la flèche ◁— .



- fonction sélectionnée : C6
- dessin d'un cycle à 6, dont on a désigné le centre avec le crayon lumineux



- fonction sélectionnée : O pour désigner des atomes d'oxygène ; on vient de désigner le 5ème .
- remarquer les liaisons en arrière (⫴⋯) et en avant (▱) du plan de l'écran, pour indiquer que l'entourage spatial du phosphore représente une bipyramide trigonale.

PLANCHE A : exemples de dessins de structures sur l'écran.

cette structure.

Les transformations sont rassemblées dans des fichiers alimentés par les chimistes d'après des recherches bibliographiques. Nous étudierons la constitution exacte et la maintenance de ces fichiers dans le paragraphe 112. Disons simplement que chaque transformation commence par un en-tête qui permet au système de décider si elle est applicable ou non à une cible donnée ; elle décrit ensuite les manipulations à effectuer sur la cible pour créer la structure du précurseur correspondant.

Le système commence par explorer complètement ces fichiers, et, pour chaque transformation, il détermine à l'aide de l'en-tête si elle est applicable ou non à la cible proposée. Dans le cas où elle l'est, on va conserver ses instructions de manipulation dans une liste dite "liste des blocs de manipulation".

Cette liste est visualisée sur l'écran sous forme d'un premier niveau de l'arbre de synthèse. Puis, pour chaque précurseur de ce niveau, le système effectue les opérations suivantes :

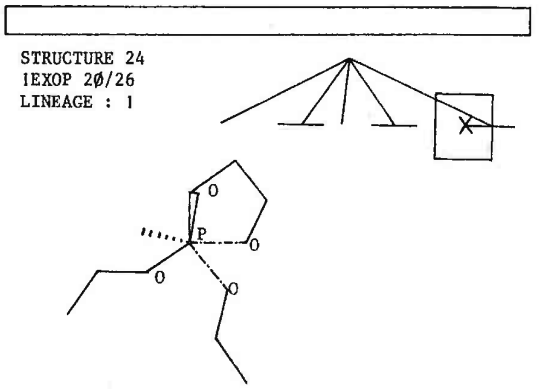
- création de sa table de connexions à partir de celle de la cible, à l'aide du bloc de manipulation
- perception chimique du précurseur
- évaluation chimique afin d'éliminer des structures identiques ou jugées chimiquement instables
- enfin, visualisation de la structure du précurseur.(Exemple planche B).

Le chimiste peut alors décider d'une étape supplémentaire, en sélectionnant comme nouvelle cible un des précurseurs déjà obtenus et en relançant le processus.

112 - Fichiers des transformations - le langage ALCHEM

Il existe six fichiers de transformations, définis sur des critères concernant en particulier le mode de sélection des transformations en fonction de la cible étudiée (on parlera du TYPE de la transformation) :

PATTRN	accès par une sous-structure prédéfinie quelconque
AROM	idem, pour les transformations traitant de la chimie aromatique
SING	accès par la présence d'un seul groupe fonctionnel



- la croix permet de pointer sur le précurseur à visualiser, avec plus de précision qu'à l'aide du crayon lumineux
- on constate que la cible peut avoir été découpée en plusieurs fragments, considérés par le système comme autant de précurseurs (traits horizontaux dans le dessin du niveau de précurseurs)
- on obtient le dessin de la molécule, avec les indications :

+ rang dans l'arbre (STRUCTURE 24), la cible étant n° 1
+ nom de la transformation créatrice, avec priorité initiale et finale
+ niveau de succession par rapport à la cible (lignée).

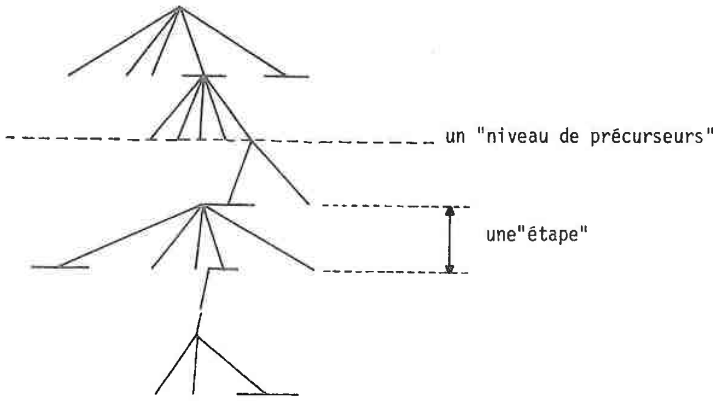


PLANCHE B : - visualisation de la structure d'un précurseur
- affichage d'un arbre à plusieurs niveaux.

PAIR accès par la présence de deux groupes fonctionnels reliés par un chemin d'atomes de carbone

INTER transformations qui échangent un groupe fonctionnel en un autre

INTRO transformations qui introduisent un nouveau groupe fonctionnel

Toutes les transformations sont définies selon le schéma général suivant :

TYPE

[références bibliographiques et commentaires

nom de la transformation

critère d'accès (sous-structure, ou groupe(s) fonctionnel(s)...)

[enquêtes structurales et conditions de réaction

[manipulations

} ces deux parties sont plus ou moins mêlées selon les cas

Les chimistes disposent pour cela d'un langage écrit spécialement pour cet usage : ALCHEM (algorithmic language for chemistry). Il permet de décrire des objets, des actions, des conditions chimiques par des phrases plus ou moins rigides exprimées à l'aide du vocabulaire chimique courant. On y trouve des instructions conditionnelles dont la structure est celle des tests d'algol (IF - THEN - ELSE - BEGIN - DONE).

Nous allons étudier brièvement les possibilités de ce langage, à partir d'exemples. Le lecteur intéressé trouvera plus de détails dans (12,13).

1) Entête de réaction

```
                TYPE PATTRN                    type (fichier choisi)
;
; EXEMPLE : REACTION DE XYZ
; REFERENCES BIBLIOGRAPHIQUES                  commentaires d'introduction
;   ET COMMENTAIRES
XYZ                                            nom de la transformation

P (-C (-C)(=O)) (=O)/                          sous-structure utilisée qui représente l'entité structurale :
                                                       P - C - C
                                                       ‖   ‖
                                                       O   O
PRIORITY 100                                   "note" attribuée au départ
```

2) même chose

```
                TYPE PAIR
;
; COMMENTAIRES...
EX-REAC-PAIR

ALCOHOL ALDEHYDE PATH 4                        cette transformation s'applique à des molécules comprenant un groupe alcool relié à un groupe aldehyde par un chemin de 4 carbones (y compris les extrémités)

PRIORITY 80
```

3) Enquêtes structurales

```
IF ATOMS ALPHA TO ATOM 3 ARE ALL CARBON THEN KILL
(la réaction est impossible si tous les atomes directement liés à l'atome 3
sont des atomes de carbone).
IF ATOM 5 IS PRIMARY THEN
   BEGIN IF OXYGEN IS ALPHA TO ATOM 1 THEN KILL
         MAKE BOND FROM ATOM 1 TO ATOM 6
   DONE
```

(la numérotation des atomes est donnée par l'ordre dans lequel ils apparaissent dans la sous-structure ou correspondent à une numérotation standard des groupes fonctionnels).

Les enquêtes permettent de tester l'environnement de la sous-structure (ou des groupes fonctionnels...) dans la molécule sur laquelle on travaille effectivement. Selon les cas, la transformation sera abandonnée, ou au contraire achevée ; on pourra également modifier PRIORITY pour traduire soit une augmentation, soit une diminution de la facilité avec laquelle la réaction s'exécute (mais PRIORITY reste une notion très subjective).

4) Conditions expérimentales

```
CONDITIONS NUCLEOPHYLIC AND HOT
CONDITIONS STRONGLY OXYDIZING
```

5) Manipulations

```
MAKE BOND FROM ATOM 1 TO ATOM 3      (créer une liaison entre les atomes
                                      1 et 3)
ADD C TO ATOM 2                      (relier un nouvel atome de carbone à
                                      l'atome 2)
BREAK BOND FROM ATOM 1 TO ATOM 2     (diminuer la liaison entre les atomes
                                      1 et 2 : double devient simple, etc...)
MODIFY VALENCE FOR ATOM 2 TO 3       (la valence de l'atome 2 devient 3)
```

6) Définition d'ensembles

On dispose de huit "set registers" dans lesquels on peut stocker des listes d'atomes (etc) possédant une certaine propriété, pour les réutiliser ensuite à d'autres endroits :

```
IF UNSATURATED ATOMS ARE ALPHA TO ATOM 5 OFFPATH (2)
```

(s'il y a des atomes non saturés directement liés à l'atome 5 hors du chemin de carbones, mettre leurs numéros dans le set register 2)
On pourrait ensuite utiliser ce set register numéro 2 :

```
IF (2) IS NITROGEN THEN KILL
IF (2) IS OXYGEN THEN ADD 20 FOR EACH
```

(on augmente PRIORITY de 20 pour chaque oxygène contenu dans (2), ce qui signifie que la réaction est d'autant plus favorisée qu'il y a d'atomes d'oxygène répondant à la condition par laquelle on a défini ce set register 2).

7) Instructions numériques

On dispose également de huit registres pouvant contenir une valeur numérique (cardinal d'un ensemble, PRIORITY...). Il est donc possible de faire des opérations arithmétiques :

```
SET VALUE 1 TO COUNT (2)
```

(le registre 1 prend pour valeur le cardinal de l'ensemble 2).

8) Exemple complet

```
TYPE PAIR
;
; S. PATAI, THE CHEMISTRY OF ALKENES, INTERSCIENCE (1964), P. 477
; MICHAEL ADDITION TO AN AMIDE
MIK/AMIDE                                          nom de réaction
AMIDEZ WGROUP PATH 3 PRIORITY 20                   { groupe 1, groupe 2
                                                   { longueur du chemin
  IF ATOM 2 IS QUATERNARY THEN KILL
  CONDITIONS SLIGHTLY NUCLEOPHILIC AND BASIC
  BREAK BOND 3 IN GROUP 1
  MAKE BOND 1
END
```

113 - Structure informatique de PASCOP

Là encore, on en fera une description assez superficielle en renvoyant à (12) pour les détails. Trois points sont particulièrement importants : la table de connexions (dans les molécules), les transformations (et les fichiers), et l'arbre. Nous allons les étudier l'un après l'autre, en se bornant à ce qu'on sera amené à utiliser dans la suite.

1131 - Représentation des molécules

La table de connexions comporte une partie relative aux atomes et une partie relative aux liaisons. Elle est de taille fixe et possède 72

entrées - atomes de 4 mots - mémoire chacune et 72 entrées - liaisons de 2 mots - mémoire ; elle occupe donc 432 mots. On constate qu'une molécule traitée par PASCOP ne pourra pas avoir plus de 72 atomes, mais ce nombre est largement suffisant, surtout vis à vis de la taille de l'écran.

Chaque entrée - atomes contient en particulier :

- un indicateur de validation
- le type de l'atome
- sa valence
- un descripteur pour la stéréochimie
- le nombre de liaisons explicites autour de cet atome
- le nombre d'atomes explicites liés
- la liste des numéros des atomes liés
- la liste des numéros des liaisons issues de l'atome considéré.

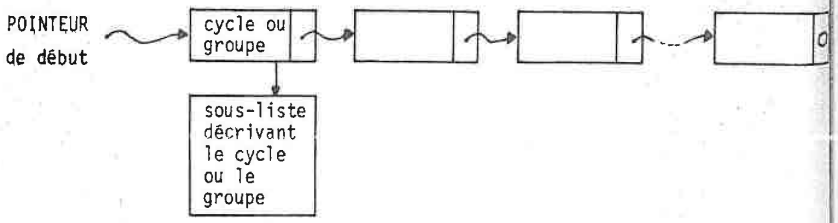
Chaque entrée - liaison contient :

- un indicateur de validation
- le type de la liaison (simple, double, triple)
- les numéros des atomes définissant la liaison
- la position de la liaison par rapport au plan de l'écran.

La numérotation des atomes est faite dans l'ordre où ils sont dessinés.

En plus de la table de connexions, on dispose de divers ensembles :

- ensembles des atomes de chaque type employé (oxygène, azote...) sous forme de chaînes de 72 bits (un double mot)
- listes des cycles et des groupes fonctionnels, sous la forme :



On constate donc que la représentation complète d'une molécule occupe une place importante en mémoire. Ceci est dû principalement au fait

que cette représentation est largement redondante. En effet, la molécule peut être considérée comme étant un graphe dont les points sont les atomes et les arcs, les liaisons. Or, on représente ce graphe par une liste des successeurs de chaque point doublée d'une liste des arcs ; de plus, les listes des cycles et des groupes fonctionnels constituent en quelque sorte un nouveau regroupement de ces informations.

Une telle redondance d'informations n'est pas gênante, puisqu'on ne fait jamais de mise à jour de la cible directement sur ses tables. Au contraire, elle met à la disposition du système un maximum de données susceptibles d'être utilisées immédiatement en cours de synthèse.

Mais on verra dans le paragraphe 1133 que la place importante nécessitée pour stocker ces informations a orienté le choix de la représentation de l'arbre de synthèse.

1132 - Les fichiers de transformations

Les instructions qui composent chaque transformation sont compilées en un code binaire plus compacte, avant d'être placées dans le fichier. Le compilateur s'appelle SYNCOM.

Chaque fichier est composé de deux zones : le "répertoire" et la zone du code généré des transformations. Pour chaque transformation, le répertoire comporte une entrée de deux mots, contenant le critère d'accès et l'adresse de la transformation elle - même dans la deuxième zone du fichier, afin d'éviter un accès systématique au disque par transformation.

Il convient de noter les points suivants :

* 1) SYNCOM ne connaît pas les transformations individuellement : il ne peut que compiler un fichier complet, considéré comme un tout. Il en résulte que l'adjonction ou la modification d'une transformation dans un fichier nécessite une re-compilation complète de tout le fichier, avec les complications et les pertes de temps que cela comporte.

* 2) la structure algorithmique des enquêtes est traitée sans aucune instruction de saut (saut - si - faux pour le THEN et saut inconditionnel avant le ELSE) : SYMCOM conserve les BEGIN et les DONE qui encadrent les blocs d'instructions correspondantes, le saut vers un ELSE ou après un bloc - THEN de l'enquête étant réalisé à l'exécution par comptage algébrique de ces BEGIN et DONE.

* 3) lors de l'exécution d'une transformation, un sous-programme établit la correspondance entre la numérotation des atomes de la cible et

celle de la sous-structure décrite dans la transformation. De plus, une même transformation peut s'exécuter plusieurs fois sur une même cible, si sa sous-structure y est retrouvée à plusieurs endroits différents (mis à part le cas de symétries signalées lors de la définition de la sous-structure). Dans ce cas, la correspondance entre les numérotations est refaite à chaque fois.(Voir planche C).

1133 - Représentation de l'arbre

Chaque structure présente dans l'arbre porte un numéro : la cible a le numéro 1, les précurseurs sont numérotés à partir de 2 dans l'ordre où ils sont crées.

On a vu que la représentation complète d'une molécule occupe une place très importante en mémoire (au minimum 432 mots). Il en résulte que déjà le système initial SECS écartait d'emblée l'éventualité de conserver la table de connexions de tous les précurseurs.

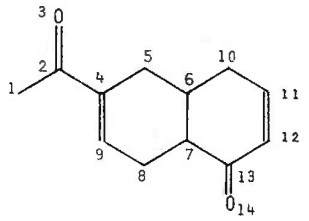
La méthode retenue est la suivante. Pour chaque molécule de l'arbre, on conserve un bloc de structure (=BS) comportant les indications suivantes (entre autres) :

- numéro d'ordre de la structure
- pointeur vers le BS du père (ou -1)
- " " " " du frère droit (ou -1)
- " " " " du fils gauche (ou -1)
- nom de la transformation créatrice
- pointeur vers le bloc d'instructions de manipulations qui permettent de passer du père à la structure considérée.

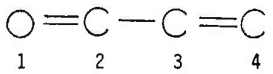
Un bloc de structure occupe 13 mots ; il n'y a en général qu'un nombre réduit de mots dans le bloc de manipulations.

On obtient par cette méthode un gain de place très appréciable permettant de conserver l'arbre complet en mémoire centrale. Par contre, la visualisation d'une structure quelconque de l'arbre sera considérablement ralentie ; en effet, il faudra pour cela effectuer les opérations suivantes :

- remonter de la structure désignée jusqu'à la cible, en retenant au passage tous les blocs de manipulations
- réordonner ces blocs dans le sens cible vers précurseur, puis les exécuter à partir de la table de connexions de la cible, qui est la seule à être toujours conservée

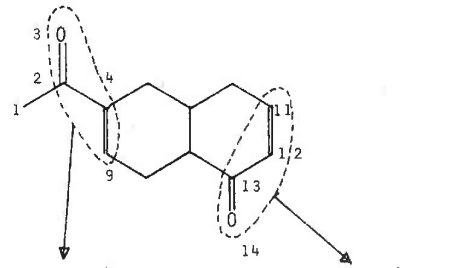


cible avec sa numérotation



sous-structure avec sa numérotation (déduite de la programmation de la transformation dans le langage ALCHEM)

elle se retrouve en 2 endroits différents :



correspondance :

1 → 3
2 → 2
3 → 4
4 → 9

correspondance :

1 → 14
2 → 13
3 → 12
4 → 11

PLANCHE C : correspondance cible / sous-structure

- on obtient ainsi la table de connexions de la structure demandée, dont on peut faire la perception et le dessin.

En fait, il convient de faire les deux constatations suivantes:

* 1) toutes ces opérations se passent en mémoire centrale, sur une unité centrale des plus rapides ; de plus, un accès au disque fait perdre l'unité centrale au profit d'un autre utilisateur. Il est donc difficile de faire une comparaison entre cette méthode et celle qui consisterait à conserver toutes les tables de connexions sur disque, mais elle n'est certainement pas moins performante.

* 2) la visualisation de précurseurs n'est pas une opération très fréquente : elle ne se fait en fait qu'à la fin de la construction de chaque niveau, à quelques exceptions près.

12 - LES LIMITES DE CE SYSTEME

Certains points de conception de ce système entraînent soit des limites effectives pour ses performances, soit des complications quant à son utilisation. Il s'agit tout particulièrement des suivants :

* 1) le compilateur SYNCOM travaille par fichiers complets, il ne distingue pas les réactions les unes des autres. Il est donc impossible de faire simplement les opérations classiques de maintenance. Notons toutefois que ce problème n'influence en aucune manière le fonctionnement du système de synthèse en lui-même, mais il apparaît comme un frein à son emploi rationnel.

* 2) le mode de compilation des enquêtes occasionne des pertes de temps à l'exécution.

* 3) l'utilisation de la structure d'ALCHEM est parfois malaisée pour les chimistes. En particulier, les utilisateurs chimistes souhaiteraient pouvoir définir des blocs logiques hors de la structure des enquêtes, avec accès par GO TO ; ainsi que des "sous-programmes" pour éviter des répétitions d'instructions à plusieurs endroits d'une transformation.

* 4) la limite de loin la plus importante est due au fait que le système ne peut travailler avec des considérations de stratégie. Il y a donc une grande perte de temps en exploration complète des fichiers de transformations, et on obtient souvent un nombre important de précurseurs non pertinents (il n'est pas rare d'en avoir plus de 50 %).

13 - LES AMELIORATIONS A APPORTER

A partir de là, on peut dresser une liste des modifications les plus prioritaires à apporter au système afin d'augmenter ses performances en synthèse et du point de vue de son utilisation par des non-informaticiens.

* 1) simplifier l'utilisation d'ALCHEM par l'adjonction d'instruction de sauts (GO TO) et de macro-instructions pour répondre au désir des utilisateurs. Cette première partie ne fait appel qu'à des techniques de compilation très classiques, je ne les décrirai pas en particulier ici. Elle aura eu pour principal mérite de me faire connaître avec plus de précisions la réalisation du système vu de l'intérieur.

Une seconde amélioration importante au niveau des fichiers consistera à permettre la compilation de transformations séparément. Nous verrons dans le chapitre II comment ce problème aura été traité avec d'autres.

* 2) Dans un premier temps, il faudrait au moins que l'utilisateur puisse communiquer au système des considérations stratégiques. La stratégie pourrait être définie soit comme un ensemble d'actions à entreprendre pour atteindre un objectif donné, soit comme un ensemble de conditions caractérisant les chemins à créer, donc les transformations à exécuter, qui permettent d'éliminer au plus tôt ceux qui ne sont pas pertinents. La recherche des transformations pourrait donc se faire en fonction de ces directives, ce qui permettrait une sélection plus rigoureuse des transformations, donc un gain de temps appréciable à l'exécution. De plus, en confrontant les précurseurs proposés avec les actions demandées, on pourrait ne conserver dans l'arbre que des précurseurs qui correspondent à la stratégie suivie par le chimiste. Enfin, le système ayant ainsi un guide, il deviendrait possible de le faire travailler sur plusieurs étapes à la fois, donc en profondeur, plutôt que toujours en largeur.

* 3) Après cela, on envisagera de ne plus laisser le chimiste seul maître des choix, le système étant capable de résoudre certains problèmes bien définis. Il pourra ainsi participer à la stratégie. Plusieurs solutions semblent dès maintenant envisageables :

- constitution "d'automates de stratégie", sortes de macro-transformations qui décrivent des chemins de synthèse connus dans des cas précis.

- comparaison de molécules ou de chemins, afin de tenir compte automatiquement de contraintes de rendements, de coût de produits, etc ; ou de pouvoir classer des chemins l'un par rapport à l'autre ; ou, enfin, de définir les produits commercialisés dont la structure est "la plus proche" de la cible étudiée.

On sera amené à associer aux chemins des fonctions de coût permettant de définir les chemins à retenir comme étant ceux qui minimisent cette fonction.

14 - CONCLUSIONS

On constate en premier lieu que le champ des recherches à élaborer est très vaste et que les améliorations qu'on peut espérer apporter sont particulièrement importantes. Mais il ne faudra pas oublier qu'il y a une structure de programme antérieure à respecter et qu'on devra y rester conforme, ne serait-ce que pour éviter une reconversion des utilisateurs non-informaticiens ; ceci conditionnera beaucoup de choix. De plus, il faut reconnaître que le système initial est relativement performant, dans le cadre de ses possibilités.

Sur un plan plus informatique, nous voyons qu'il s'agit d'un problème d'intelligence artificielle, puisqu'on cherche à automatiser une démarche humaine. En fait, dans les premiers développements de la stratégie, nous avons vu que l'initiative reste au chimiste puisqu'il se contentera de communiquer ses désirs. Cette première amélioration permettra surtout de servir de tremplin pour les suivantes. A ce niveau, on est plutôt en face d'un problème de cheminement dans un graphe particulier :

- on ne connaît que la racine (la cible), les autres points du graphe étant déterminés au fur et à mesure par la construction des arcs
- le nombre de points est impossible à chiffrer, mais relativement très grand
- chaque arc peut être long à déterminer

- on ne connaît pas (habituellement) les points d'arrivée, mais on a des renseignements sur la construction des chemins.

Il résulte de tout ceci que si le problème reste original, on ne pourra pas lui appliquer tels quels les résultats classiques de la théorie des graphes finis.

Une fois définie cette base permettant le guidage du cheminement dans ce graphe (l'arbre de synthèse) depuis l'extérieur, il sera possible d'y greffer des modules assurant le guidage automatique : recherche de chemins de coût minimal, ou de chemins passant par un point ou un arc donné,... A ce moment, on entrera plus avant dans le domaine de l'intelligence artificielle.

Un autre problème se trouvera posé : celui de la communication entre l'utilisateur non-informaticien et le système. On aura à définir un "langage" qui permette d'exprimer des opérations sur des objets de l'univers des chimistes et utilisé par des personnes qui ne devraient pas avoir à faire un long apprentissage pour s'en servir. Les traitements à définir doivent l'être en restant dans cet univers. Il s'agit donc d'un langage orienté "problème" et non d'un langage de programmation classique.

De plus, on rencontrera des problèmes de recherche d'articles dans un fichier en fonction de nombreux critères, le délai de réponse devant être aussi bref que possible, tout se passant en conversationnel. On sera donc amené à employer des techniques analogues à celles qu'on utilise dans les bases de données.

On voit ainsi que l'élaboration de ce système amènera à des développements de techniques variées dans le domaine de l'informatique tout en contribuant à la recherche avancée en chimie.

CHAPITRE II

LA STRATEGIE

Les améliorations que l'on désire apporter au système doivent permettre de diriger la construction de l'arbre de synthèse, au lieu de le laisser se développer au maximum en largeur. Or on a vu que les principaux moyens de guidage et de recherche employés par les chimistes lorsqu'ils travaillent sans l'aide de l'ordinateur sont principalement d'une part,leurs connaissances, leur intuition et leur expérience,et d'autre part des critères économiques basés sur l'existence et le prix (etc) de produits.

On conçoit aisément que ces deux aspects vont donner lieu à des développements complémentaires, mais de nature très différente. En effet, si les critères économiques sont faciles à définir en théorie, il n'en est pas de même pour "l'intuition, l'expérience et les connaissances" d'un homme.

Il serait faux d'en conclure que l'approche la plus simple sera de dresser le catalogue des produits commercialisés classés par ordre de prix de revient croissant (par exemple). Un tel procédé mènera très vite aux problèmes d'intuition puisqu'il faudra de toutes façons pouvoir comparer non seulement l'égalité d'une molécule donnée avec une du catalogue, mais aussi la "différence de complexité" entre ces molécules. De plus, il faut être conscient du nombre de produits commercialisés : le catalogue de la Société ALDRICH (15) qui semble le plus complet du marché, en propose environ 10 000 ! On est toutefois ici en présence d'un moyen de guidage à ne pas négliger dans le futur.

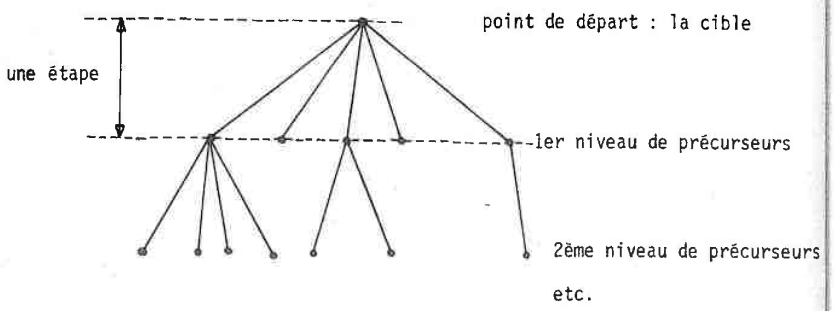
On peut imaginer au contraire une méthode plus simple qui permettrait déjà un développement important : laissons l'intuition et l'expérience du chimiste aux chimistes et donnons leur simplement un outil rapide

de test, assorti d'une solide "connaissance bibliographique". Dans ce procédé, le système informatique ne fait pas de stratégie, il se comporte comme un système de documentation aussi puissant que possible, qui propose à l'utilisateur diverses solutions puisées dans ses "connaissances" en fonction des indications données de l'extérieur. Néanmoins, on a là aussi un moyen capable de diriger le cheminement dans l'arbre de synthèse,en particulier de réduire la largeur de chaque niveau obtenu.

On peut conclure de cela que les domaines et les directions qu'il faudra étudier seront assez variés. En particulier, on devra mettre au point des procédures où le système ne fait pas de stratégie, et d'autres plus automatiques, où il devra participer à la stratégie de synthèse.

21 - PROBLEME INFORMATIQUE GENERAL

Schématiquement, l'arbre de synthèse a la structure suivante :



Chaque niveau de précurseur est un ensemble dont les éléments sont des molécules ; la cible peut être considérée comme un niveau zéro, à un seul élément $P_{0,1}$. Il y a une relation entre les éléments d'un niveau et ceux du niveau suivant, qu'on peut définir ainsi :

le précurseur $P_{i,j}$ du niveau i est relié au précurseur $P_{i+1,k}$ du niveau i+1 si et seulement s'il existe une transformation qui permet de passer de $P_{i,j}$ à $P_{i+1,k}$.

Rappelons que tout au long de la recherche de chemins de synthèse, on s'intéressera aux transformations (définissant les modifications dans le sens rétro-synthétique), alors qu'une fois un chemin défini, l'expérimentateur le remontera vers la cible en essayant d'appliquer les réactions correspondantes (dans le sens synthétique). Or, par construction, il y a une bijection entre l'ensemble des réactions et celui des transformations. On peut donc bien décrire la relation entre précurseurs dans le sens rétro-synthétique comme cela a été fait ci-dessus.

On conclut de tout ce qui précède que l'arbre de synthèse est le graphe (M,t) où M est l'ensemble des molécules et t la relation "transformation" définie par :

$\forall P, Q \in M, \quad P\ t\ Q \iff$ il existe une transformation qui fait passer de P à Q (donc une réaction qui fait passer de Q à P).

On appelle habituellement en théorie des graphes (16) "successeurs de P", noté t(P), l'ensemble des points Q du graphe tels que PtQ. On remarque que les successeurs de P sont en fait ses précurseurs au sens chimique. Il conviendra donc de faire attention à la confusion entre ces termes. Dans la suite, nous n'employerons que le mot précurseur.

D'autre part, il existe a priori une infinité de molécules différentes. On risque donc d'avoir des difficultés pour appliquer les résultats classiques de la théorie des graphes finis. Par ailleurs, on ne peut pas définir au départ un ensemble de points plus réduit, car dans la majorité des cas, l'expérience du chimiste ne lui permet que d'avoir certaines présomptions sur des caractéristiques que pourraient avoir des précurseurs pertinents. De là, on conçoit qu'il sera peut-être possible de définir un sous-ensemble (et non tout l'ensemble) des points du graphe en compréhension, mais très probablement pas en extension. En effet, le plus souvent, la donnée d'une propriété déterminera non pas une molécule précise, mais un ensemble de molécules, donc de points du graphe.

Le graphe (M,t) ne peut donc pas être représenté par la liste de ses points, ni par une liste d'arcs, mais par une liste de "générateurs" d'arcs : une transformation est un programme permettant de construire des arcs du graphe. De plus, cette opération de construction d'arc est relativement coûteuse : recherche de sous-structure, enquêtes, manpulations, perception et évaluation.

La recherche de chemins d'origine donnée dans ce graphe va

donc consister à construire le sous-graphe contenant les points pouvant être atteints depuis cette origine. Il sera représenté par un arbre considéré comme équivalent à une traduction près.

On sera donc amené à chercher des moyens originaux, adaptés à ce problème en particulier. Mais on n'ignorera pas pour autant les résultats de la théorie des graphes finis. Les cas qui semblent les plus intéressants sont les suivants :

- recherche de tous les chemins (c'est presque le fonctionnement actuel)
- recherche d'un seul chemin
- " d'un chemin formé d'un minimum d'arcs
- " " " de coût ou de rendement minimal
- " de quelques chemins répondant à un des critères précédents (coût, rendement ou nombre d'arcs limité)
- recherche de chemins passant par ou aboutissant à un point donné, ou passant par un arc donné.

Par contre, on notera qu'on ne s'intéresse absolument pas aux circuits. On ne veut trouver que des chemins élémentaires.

En conclusion de cet énoncé général, nous dirons que pour aborder les problèmes dans un ordre de complexité croissante, nous commencerons par développer des méthodes "manuelles" et "semi-automatiques". Nous amorcerons ensuite l'étude de procédés plus automatiques. Mais leur réalisation pratique ne sera faite que lorsque les chimistes auront pu employer les premiers procédés, de façon qu'ils puissent définir des besoins plus intéressants et plus efficaces au vu de leurs expériences. De plus, cette première phase permettra des observations plus précises sur les comportements du système d'une part (mesures de temps, etc...) et des utilisateurs d'autre part.

22 - STRATEGIE FAITE PAR LE CHIMISTE : "STRATEGIE INTERACTIVE"

Comme par le passé, le chimiste continue à faire la recherche stratégique lui-même . En particulier, c'est toujours lui qui choisit après chaque niveau le précurseur à partir duquel il veut continuer. La différence

fondamentale est due au fait que maintenant, il communique avant toute exécution ses désiderata au système : il doit pouvoir lui signaler par exemple que certaines parties de la molécule doivent rester inchangées, que telle autre doit subir telle modification, etc...

A partir de là, il faut pouvoir limiter le temps de recherche dans le fichier des transformations,et le nombre de précurseurs non pertinents à chaque niveau.

Le système qui fonctionne actuellement en recherchant tous les arcs ayant une origine donnée (t(x)) sera donc modifié pour ne plus fournir que les arcs (xy) de t(x) tels que y possède certaines propriétés structurales données. En somme, on peut considérer que les indications de stratégie fournies au système par l'utilisateur permettent de définir le graphe comme étant en réalité un multi-graphe dans lequel la relation s définie par

$P \; s \; Q \iff$ il existe un moyen de passer de P à Q par une transformation

est en fait une combinaison de relations

$$s = s_1 \; o \; s_2 \; o \ldots$$

où o est un opérateur ET, OU inclusif ou bien OU exclusif, et où chaque s_i est de la forme

$P \; s_i \; Q \iff$ il existe une transformation qui fait passer de P à Q en vérifiant les conditions demandées par la ième directive de stratégie.

L'ancien fonctionnement du système devient simplement le cas particulier du nouveau où l'ensemble des indications préalables données par l'utilisateur est vide.

Un premier problème apparaît tout de suite : celui de la communication chimiste - système. En effet, il faudra pouvoir fournir au système des directives précises qui lui indiquent les opérations chimiques que l'on voudrait pouvoir faire sur la cible courante, c'est à dire définir la nouvelle relation s . De plus, ces directives doivent pouvoir être combinées entre elles pour qu'on puisse définir plusieurs actions à entreprendre simultanément, en une étape ou en plusieurs étapes, ou encore plusieurs actions équivalentes... On définira donc des combinaisons logiques de directives

(que nous appellerons CLD dans la suite) où les directives sont reliées par les opérateurs booléens classiques ET, OU inclusif, OU exclusif et négation.

Les questions soulevées par les moyens à mettre en oeuvre pour résoudre ces problèmes de communication entre un utilisateur non-informaticien et l'ordinateur sont des questions importantes. En effet, des choix effectués dépendront de beaucoup les possibilités d'emploi du système : il est peu efficace de présenter un système performant et puissant mais d'un emploi trop compliqué pour être apprécié à sa juste valeur. De plus, on peut penser dès maintenant, et la suite de ce chapitre va le confirmer, que quels que soient les nouveaux développements apportés au système, l'utilisateur aura à établir une communication accrue avec lui. Il en résulte que l'on retrouvera ce problème à tous les niveaux du développement du système. On en fera donc un chapitre à part : dans le chapitre III, nous étudierons la manière dont nous avons répondu à toutes ces questions, avec les motivations de ces choix ; quant au chapitre IV, nous y exposerons le langage défini, vu par l'utilisateur. On pourra ainsi apprécier dans ce chapitre somme toute assez peu informatique, l'étendue, la souplesse et les possibilités offertes à des utilisateurs qui doivent pouvoir employer le système sans être programmeurs.

Pour la suite du chapitre II, nous considérerons simplement qu'il existe un "langage" qui permet :

- de réaliser des enquêtes structurales sur la molécule ou des tests de valeurs numériques
- de donner des instructions générales au système (calculs, définition de limites de coût ou de rendement à respecter, définition d'une partie de la cible à ne pas modifier, etc...)
- de lui communiquer des directives (seules ou en CLD) définissant les modifications que l'on aimerait apporter à la structure de la cible en cours d'étude.

L'une des premières conditions à respecter sera bien sûr de définir ce langage de telle sorte qu'il soit aussi facile que possible de lui apporter des modifications, de lui ajouter de nouvelles instructions ou directives, etc... En effet, la recherche dans le domaine de la stratégie va se développer bien après que soit déterminée la structure informatique du langage de communication.

L'étape suivante consiste à faire le lien entre les directives et la construction des arcs du graphe. On a vu qu'un arc se construit chaque fois qu'une transformation est applicable sur la cible, et que le précurseur auquel elle conduit est en accord avec les directives données. Une première solution serait donc de vérifier si le précurseur obtenu est valide. Mais on voit aisément qu'on n'y gagne rien en temps d'exécution : on obtiendra bien à la fin de chaque étape seulement des précurseurs qui vérifient la propriété demandée, mais on aura quand même fait, pour chaque transformation applicable à la cible, la chaîne d'exécution recherche de sous-structure / enquêtes / manipulation / perception / évaluation. Il ne faut pas oublier non plus que la recherche de sous-structure doit être faite pour toutes les transformations du fichier.

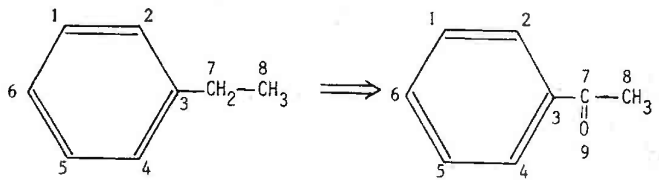
Il serait donc intéressant de pouvoir prendre la décision d'abandonner des transformations avant que leur exécution complète soit terminée, ou mieux, avant même de la commencer.

Une solution très simple consiste à faire correspondre à chaque directive donnée, l'ensemble des transformations susceptibles de l'exécuter. De cette façon, on pourra ne faire la recherche de sous-structures que sur un sous-ensemble des transformations du fichier, les autres étant ignorées. Comme on veut en plus pouvoir donner plusieurs directives reliées entre elles par AND, OR, XOR ou NOT, il faut avoir une représentation de ces ensembles sur laquelle il soit simple et rapide de répercuter ces combinaisons logiques. On est donc face à un problème de recherche dans un fichier en fonction de nombreux critères. Un ouvrage tel que (17) nous sera utile pour l'étude de ce problème.

Une deuxième solution pourra encore accélérer le processus : on peut voir assez facilement que l'exécution de certaines directives simples peut être contrôlée dès la phase "manipulation". Par exemple, si l'on demande de "casser la liaison entre les atomes 2 et 5", on peut vérifier, une fois une transformation acceptée (sous-structure correspondante et enquêtes réalisées avec succès) qu'il existe bien une instruction de manipulation qui casse la liaison demandée. On évite ainsi les phases perception et évaluation.

Il faut cependant noter que certaines directives ne pourront être vérifiées que tout en fin d'exécution ; ainsi par exemple, si l'on veut introduire un groupe cétone($\rangle C = O$) au niveau de l'atome 7 de la molécule $\underset{\sim}{I}$

suivant le schéma :



1 ∿ 2 ∿

la bonne implantation du groupe cétone ne pourra être vérifiée qu'en construisant la liste de l'ensemble des groupes fonctionnels de 2 ∿ .

On aura donc principalement deux problèmes à traiter :

- la recherche multicritères dans le fichier des transformations
- la vérification de l'exécution de l'objectif qu'on veut atteindre.

A ce niveau, il n'y a pas de problème particulier de cheminement dans l'arbre de synthèse, vu qu'il est toujours réglé de l'extérieur. En fait, seule la largeur de chaque niveau et leur vitesse de constitution est modifiée.

221 - Recherche dans le fichier des transformations

On voudrait pouvoir faire correspondre facilement à chaque directive l'ensemble des transformations du fichier susceptibles de l'exécuter. De plus, il faut pouvoir refléter sur ces ensembles les opérations logiques qui sont susceptibles de relier les directives entre elles. Signalons qu'on travaillera sur la base d'un fichier de 1 000 transformations au plus.

2211 - La première chose à faire semble donc d'étudier la structure de ces opérations logiques. Il est probable, de l'avis même de chimistes, que la plupart du temps, les combinaisons logiques de directives auront une structure assez simple. Par exemple :

- exécuter A et B et C, où A, B et C sont des directives quelconques, éventuellement employées avec négation

- exécuter A ou B ou C
- exécuter (A et B) ou (C et D)
- exécuter A ou exclusif B.

Mais il semble que dans certains cas, la combinaison pourra être nettement plus complexe. En particulier, il ne serait pas raisonnable de ne pas permettre l'emploi de parenthèses.

Au premier abord, on pourrait simplement considérer la CLD comme une expression booléenne dont les variables sont les directives. Mais l'exécution de la CLD ne se résume pas à l'exécution de l'expression booléenne : chaque directive présente nécessite elle aussi une exécution. Cette dernière opération consiste à associer à la directive l'ensemble des transformations du fichier susceptibles de l'exécuter, puis à essayer ces transformations afin de créer les précurseurs. Il en résulte qu'on est plutôt en présence d'une expression précisant des opérations sur des ensembles : l'expression A et (B ou C) est associée à $E(A) \cap (E(B) \cup E(C))$, où E(A) est l'ensemble de transformations associé à la directive A.

Il semble donc plus avantageux d'exécuter d'abord cette expression ensembliste, puis, à partir de son résultat, les transformations retenues. Pourtant, cette méthode ne sera pas retenue, pour deux raisons principales.

En premier lieu, elle pose le problème de la représentation de ces ensembles au cours de l'exécution de l'expression. En effet, il s'agit d'une expression parenthésée, dont l'exécution nécessite une pile (18), dans laquelle il faudrait empiler au fur et à mesure la représentation de ces ensembles. Or, la dimension retenue pour le fichier des transformations est de l'ordre de 1 000 ; quelle que soit la méthode choisie pour représenter les ensembles, l'exécution de l'expression nécessitera une occupation mémoire importante.

De plus, on verra que dans certains cas, on sera amené à demander la compilation d'un programme de stratégie sans avoir fourni la cible (cas d'automates de stratégie par exemple). Il ne sera donc pas possible de définir les ensembles associés aux directives dans ces cas, ce qui revient à rejeter la totalité du travail à l'exécution, pratiquement rien ne pouvant être fait à la compilation.

C'est pourquoi il a semblé préférable de se tourner vers la méthode suivante. Considérons la CLD simple "A ou B" ; elle signifie

que l'objectif qu'on se fixe est atteint si la directive A est vérifiée, et qu'il l'est aussi si B est vérifiée. On définit donc à l'aide du OU inclusif des sous-objectifs indépendants à l'intérieur de l'objectif décrit par la CLD.

Par contre, lorsqu'on demande "A et B", on veut trouver des transformations qui génèrent des précurseurs où A et B sont simultanément vérifiées.

Quant au OU exclusif, on sait que par définition, il se ramène à des ET et des OU inclusifs :

A XOR B = (A ET non B) OU (non A ET B).

Il en résulte que seul l'opérateur ET nécessite réellement une opération sur les ensembles associés, l'opérateur OU inclusif résultant simplement en un découpage de l'objectif décrit en sous-objectifs.

Partant de ce principe, le compilateur va commencer par transformer l'expression logique donnée sous une forme disjonctive équivalente. L'objectif serait ainsi transformé en une succession de sous-objectifs distincts reliés entre eux par des OU inclusifs, au sein desquels les directives ne seraient plus reliées que par des opérateurs ET. Les seules opérations à exécuter réellement sur les ensembles associés seraient ainsi les réunions correspondantes. On essayera bien sûr de passer à la forme disjonctive minimale pour éviter les redondances qui ne sont que pertes de temps.

Exemple :

la CLD (A ou B) et (non A ou C) sera convertie en
(A et C) ou (non A et B) ou (B et C) ;

les trois sous-objectifs ainsi définis pourront être étudiés l'un après l'autre

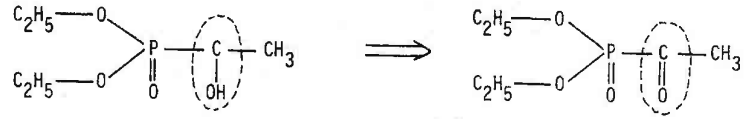
On a bien résolu de cette façon les problèmes qu'on s'était posés puisqu'on a supprimé les parenthésages et le mélange des opérateurs logiques, vu qu'il ne reste plus en fait que des ET, ce travail pouvant être fait même si la cible n'est pas connue.

On a déjà dit qu'on voudrait pouvoir demander d'exécuter une (ou plusieurs) directive(s) à l'exclusion de toute autre modification sur la molécule. Par exemple :

"échanger le groupe alcool sur l'atome 5 seulement"

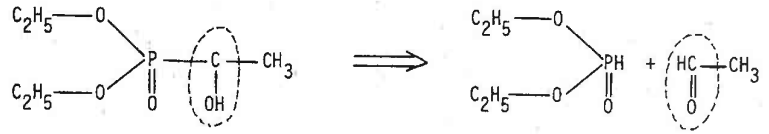
signifie qu'on veut remplacer ce groupe alcool par un autre groupe fonctionnel,

mais qu'aucun autre changement ne doit être fait :



Alcool est remplacé par Oxo "seulement"

alors que :

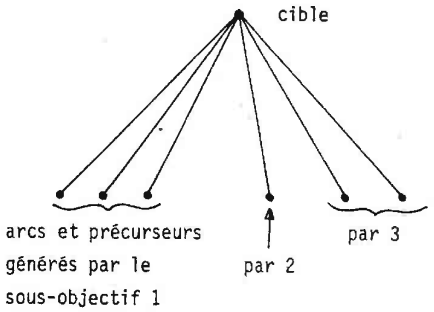


le groupe alcool est à nouveau remplacé par Oxo,
mais il y a eu un autre changement ; une telle transformation sera rejetée si l'on spécifie "seulement".

Cela signifie que si n directives sont employées avec ou sans négation dans un sous-objectif (monôme de la forme disjonctive minimale) où l'on a donné cette option, il faut que toutes les autres directives soient considérées comme employées avec négation pour traduire le fait que les changements qui leur correspondent sont interdits.

Exemple : si l'on a la forme minimale suivante :
(A et B seulement) ou (A et C), elle équivaut à :
(A et B et non C et non (toutes les autres)) ou
(A et (B indifférente) et C et (toutes les autres indifférentes)).

A partir de maintenant, nous pourrons raisonner sur un sous-objectif, l'ensemble des précurseurs possibles étant la réunion des sous-ensembles associés à chaque sous-objectif.

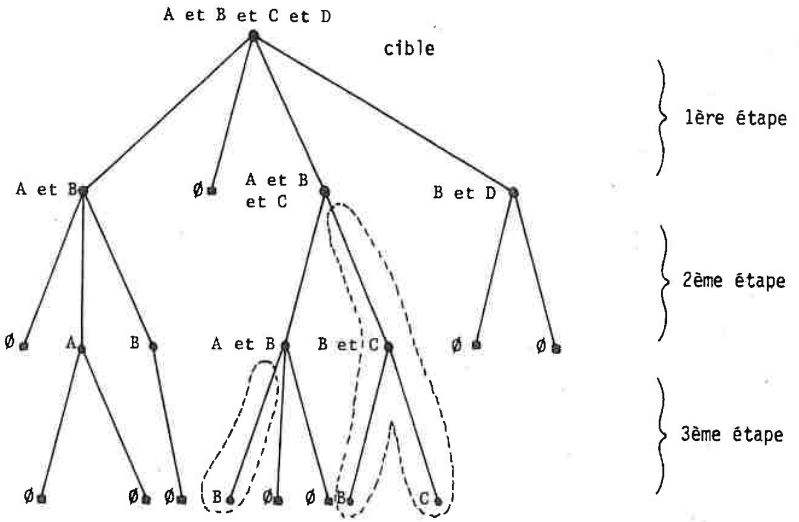


On verra au chapitre III comment on fait le passage de la CLD donnée par l'utilisateur à sa forme disjonctive minimale et comment on représente cette sorte de "table de vérité" en mémoire. On trouvera enfin au chapitre IV la syntaxe adoptée pour l'écriture des CLD.

2212 - Voyons maintenant comment s'exécute un sous-objectif, dans lequel les directives sont reliées entre elles par des opérateurs ET. Il faut faire correspondre à cette conjonction de directives l'ensemble des transformations susceptibles de l'exécuter. Deux cas sont à considérer :

- le sous-objectif doit être atteint en une seule étape. On veut donc trouver l'ensemble des précurseurs où toutes les directives du sous-objectif sont vérifiées simultanément après l'exécution d'une seule transformation. Il faut donc déterminer le sous-ensemble du fichier des transformations qui contient celles qui sont susceptibles chacune d'exécuter toutes ces directives à la fois.

- le sous-objectif doit être atteint après un nombre maximal d'étapes n supérieur à 1. On ne construit donc pas un arc unique ayant pour origine la cible, mais des chemins de longueur maximale n. Par exemple, si le sous-objectif est "A et B et C et D", avec n = 3, on pourra obtenir un arbre de ce type (en notant à côté des précurseurs les directives qui restent à exécuter) :



On constate sur ce schéma qu'il peut y avoir des chemins qui mènent à des précurseurs dans lesquels le sous-objectif n'est pas atteint au bout des n étapes ; il faudra supprimer de tels chemins (entourés en pointillés sur le schéma).

De plus, il faut maintenant trouver l'ensemble des suites de n transformations au plus susceptibles de faire atteindre le sous-objectif. Pour cela, on déterminera pour chaque structure étudiée (la cible étant la première) le sous-ensemble du fichier des transformations contenant celles qui sont susceptibles chacunes d'exécuter au moins l'une des directives non encore exécutées. Comme dans le schéma ci-dessus, il faudra conserver au niveau de chaque précurseur la liste des directives qui restent à traiter à partir de ce précurseur.

Notons que selon les chimistes, il est vraisemblable que le nombre d'étapes maximal sera généralement 1. Le cas où l'on permet un nombre d'étapes supérieur à 1 sera surtout intéressant dans l'optique du système participant activement à la stratégie (par exemple, s'il peut évaluer les chemins qui se construisent, etc...) ; on étudiera ces cas plus tard. Il n'empêche pas qu'il faut laisser à l'utilisateur la possibilité de faire travailler le système sur plusieurs étapes, même sans qu'il participe à la stratégie.

2213 - Un dernier critère à définir est celui des diverses conditions d'utilisation du fichier des transformations.

Au cours d'une recherche de chemins de synthèse, le fichier sera employé uniquement en consultation, fréquemment, et en conversationnel.

Les mises à jour de ce fichier se feront uniquement hors du système de synthèse assistée. Elles correspondent à l'adjonction de nouvelles transformations ou à des modifications de transformations déjà existantes, mais incomplètes ou erronées. Ces mises à jour seront donc plutôt faites en batch. De plus, elles seront en nombre plus limité. Il en résulte que l'organisation du fichier devra surtout favoriser les opérations de recherches par rapport aux opérations de mises-à-jour.

2214 - Conclusion : présentation du système de recherche dans le fichier qui a été retenu

On voudrait pouvoir représenter les sous-ensembles correspondant à chaque directive. La première opération à faire consiste donc à avoir un moyen de décider auxquels de ces sous-ensembles appartient chaque transformation. Il est clair que la donnée des instructions de manipulation d'une transformation ne suffit pas ; en effet, une réaction qui permet d'introduire un nouveau groupe fonctionnel,par exemple,comprendra plusieurs instructions du type MAKE BOND, mais elle ne sera pourtant pas à mettre dans le sous-ensemble correspondant à la directive MAKE BOND car ce n'est pas sa fonction chimique.

On a donc rajouté une instruction nouvelle à ALCHEM : l'instruction CHARACTERS. Notons que cette instruction avait déjà été prévue dans la version initiale du système, mais qu'elle y était toujours restée sans emploi. Grâce à cette instruction, placée en tête de transformation, le compilateur SYNCOM pourra décider des sous-ensembles concernés. Néanmoins, un problème important est apparu à ce sujet : le fait de faire se correspondre exactement les caractères et les directives rendrait l'emploi de cette instruction très délicat, et on risquerait de nombreuses erreurs d'interprétation. Prenons par exemple la cas de la directive "casser la liaison entre les atomes n_1 et n_2" Si cette liaison est sur un cycle, elle pourra être cassée par des transformati de type "casse une liaison" ou de type "casse un cycle". Il faudrait donc que l'utilisateur indique systématiquement les caractères "casse une liaison" et "casse un cycle" pour toutes les transformations de ce type. Il existe en plus

d'autres cas du même genre.

On a donc préféré la solution suivante : on a défini un certain nombre de caractères, si possible calqués sur les directives, et on représente les ensembles correspondant à ces caractères et non pas aux directives. C'est le programme d'exécution des directives qui se chargera de regrouper ces ensembles-caractères pour en faire les ensembles-directives si besoin est. L'utilisateur n'aura donc pas à se soucier de ces cas particuliers, ni à l'écriture des CHARACTERS, ni lors de la stratégie. Notons que le regroupement ensemble-caractères à ensemble-directives aurait pu être fait dès la compilation des transformations par SYNCOM. On y aurait gagné le temps correspondant lors de l'exécution d'un programme de stratégie interactive. On a refusé cette solution pour deux raisons principales :

- pour que la maintenance "chimique" du fichier des transformations reste pratique, on fournira des programmes de service à cette fin. Parmi eux, il y aura un programme qui donne la liste des transformations de l'un des ensembles donnés, donc des transformations qui possèdent un CHARACTER donné, ceci pour pouvoir faire certaines vérifications sur la valeur chimique du fichier. On doit donc conserver la représentation des ensembles-caractères. Il serait bien sûr possible d'avoir les deux, mais le rapport utilité/place ne le justifie guère.

- de plus, on verra dans la suite de l'étude, qu'on rapporterait sur chaque compilation un temps de travail probablement supérieur à celui qu'on passera dans certaines exécutions de synthèses.

Il reste donc à choisir une méthode de représentation de ces ensembles. Une étude détaillée de plusieurs possibilités a été faite d'après les résultats de (17) sur les fichiers inversés multi-critères, conjointement avec M. PFAADT (21). Nous n'en reprendrons ici que les points les plus importants en renvoyant au rapport de PFAADT pour les détails.

La représentation choisie doit donc permettre une consultation rapide en conversationnel à partir d'opérations logiques faites sur un certain nombre de critères (directives) pris parmi un nombre relativement important. On verra en annexe C qu'on en dénombre actuellement 68 dont cinq peuvent se décomposer en sous-critères.

L'étude faite par M. PFAADT consistait en particulier à comparer diverses méthodes pour inverser le fichier : représentation par chaînes de bits, fichier inversé sur plusieurs critères, fichiers multicritères in-

versés. On en a conclu que la seule méthode qui puisse allier rapidité et occupation mémoire et disque raisonnable est la représentation par chaînes de bits. C'est de plus, la plus simple, ce qui est un avantage loin d'être négligeable.

On va donc considérer que le fichier est composé de F articles numérotés de 1 à F. Pour chaque caractère, on représente l'ensemble des articles possédant ce caractère par une chaîne de F bits où le bit i est à 1 si et seulement si la i ème transformation en fait partie.

On conçoit aisément que cette méthode permettra de réaliser très simplement des opérations logiques sur ces ensembles : il suffira de faire les mêmes opérations sur les chaînes de bits.

Or, on a vu que, dans l'ancienne version du système, les transformations sont précédées par un "répertoire" comportant un descripteur par transformation, et que le code de transformation elle-même est de taille variable.

On a vu au chapitre I l'utilité de ce répertoire ; l'introduction de la stratégie interactive ne modifie en rien ce principe. On conserve donc le répertoire. De plus, les descripteurs étant de taille fixe, on passe du numéro d'une transformation à l'adresse relative de son descripteur par adressage calculé (22) dont la fonction est une simple multiplication par une constante. De là, le descripteur peut contenir l'adresse réelle sur disque du code de la transformation (adressage par chaînage). Il n'y a donc aucun problème jusqu'ici à maintenir les codes de transformation de taille variable.

Il faut toutefois considérer les cas suivants :

* 1) accès à une transformation par son nom : il suffit d'avoir une table de correspondance nom-numéro et/ou nom-adresse du code.

* 2) mise à jour : elle serait évidemment simplifiée si le code des transformations occupait une taille fixe. Mais une statistique sur l'état actuel du fichier donne les résultats suivants :

⊕ si l'on donne à la taille d'un bloc la taille de la plus grande transformation, il faudrait des blocs de six secteurs (168 mots) et on aurait une perte de place d'environ 23 % de la taille totale du fichier.

⊕ si l'on donne à la taille d'un bloc la taille moyenne des transformations arrondie au nombre entier de secteurs immédiatement supérieur,

il suffirait de blocs de 2 secteurs. On aurait alors une perte de seulement 10 % de place et 12 % des transformations seulement nécessiteraient plus d'un bloc.

⊕ en mettant simplement les transformations sur un nombre entier de secteurs, la perte de place devient négligeable (< 3 %), et la répartition se fait de la façon suivante :

1	secteur	environ 40 %
2	secteurs	" 48 %
3	"	" 8 %
4	"	" 2 %
5	"	" 1,5 %
6	"	" 0,5 %

⊕ une autre méthode consisterait à allouer à chaque transformation un bloc de taille fixe contenant son descripteur suivi du début du code de la transformation (ou de tout le code s'il loge) . Il serait ainsi possible d'atteindre le descripteur et le début du code par adressage calculé, la suite de code étant atteinte par chaînage. Les chiffres précédents montrent que pour que cette méthode soit rentable, il faut que ce bloc initial ait une longueur d'une trentaine de mots au moins ($\sim$ 1 secteur, 40 % des transformations n'auraient pas de suite chaînée). Il en résulte qu'on ne pourrait plus charger le répertoire complet en mémoire, donc que son balayage nécessiterait plusieurs accès. De plus, les opérations de mise à jour ne seraient pas simplifiées.

On a conlu de ces résultats que la place perdue en prenant des blocs de taille fixe n'est pas justifiée vis à vis de l'apport de simplicité, d'autant plus que peu de transformations sont vraiment longues. Par contre, on devra ajouter une chaîne de bits d'occupation qui permettra de noter l'emplacement de trous laissés par des mises-à-jour avec augmentation de taille.

Jusqu'ici, on a parlé "du" fichier des transformations ; or, on se souvient que dans l'ancienne version, il s'agissait en fait des six fichiers PATTRN, PAIR, SING, INTER, INTRO et AROM. On a employé le singulier parce que les problèmes de ces six fichiers étaient identiques; de plus, on constate qu'il est maintenant possible de les regrouper physiquement, la différence

étant faite par six chaînes de bits.

Enfin, notons qu'on a jugé préférable de donner la possibilité d'accéder à une transformation directement sans passer par le répertoire. Il faut donc une table de correspondance numéro de transformation - adresse.

A partir de tout cela, on arrive à la structure décrite sur la planche D.

Résumé :

1) accès par les chaînes de bits

- le rang i d'un bit à 1 de la chaîne considérée est multiplié par la longueur d'un descripteur (= 2 mots)
- on connaît ainsi le descripteur de la réaction i : sous-structure sur laquelle elle travaille, adresse du code et type
- si besoin est (la sous-structure correspond) on peut aller chercher la transformation dans le fichier, à l'adresse trouvée dans le descripteur, et commencer à l'exécuter.

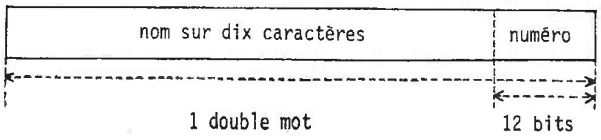
2) accès par le numéro de réaction

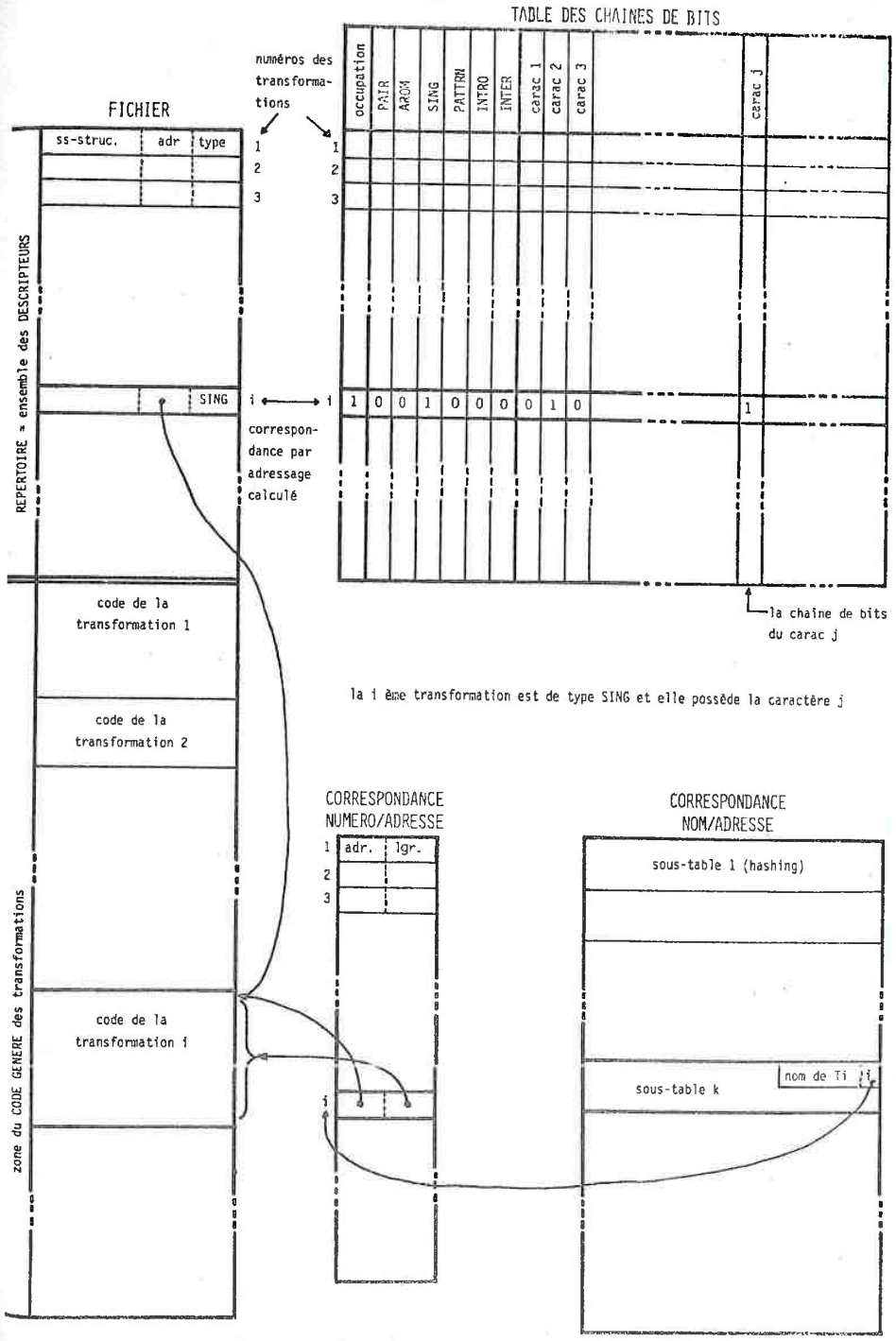
Il s'agit d'un cas particulier de l'accès par les chaînes de bits qu'on n'utilisera que si on n'a qu'une transformation à essayer dont on connaît le numéro :

- le numéro i sert à indicer le tableau de correspondance entre les numéros et les adresses : on trouve dans la i ème entrée l'adresse et la longueur du code de la transformation i
- on peut donc la chercher dans le fichier à l'adresse donnée.

3) accès par le nom de la transformation

L'ancienne version acceptait des noms de transformation dont seuls les cinq premiers caractères étaient pris en considération. Ce nombre semblant trop petit,on l'a augmenté tout en respectant une taille compatible avec la constitution d'une table des noms des transformations : en prenant 10 caractères, on pourra définir chaque entrée de cette table de la façon suivante :





Il restait à organiser cette table de façon que la recherche soit suffisamment rapide. On a opté pour une solution par découpage en sous-tables à l'aide d'un "hashing" qui combine une rapidité suffisante pour les besoins et une occupation mémoire assez faible :

- on prend les six premiers caractères du nom à rechercher (un mot mémoire), on divise cette valeur binaire par un nombre premier tel que les sous-tables puissent contenir un nombre restreint de noms, et on conserve le reste de cette division. On a ainsi le numéro (moins 1) de la sous-table à employer. Pour une table de mille noms au maximum, on a choisi 203 comme diviseur ; la moyenne d'occupation des sous-tables est donc de 5 noms environ. On a prévu les sous-tables suffisamment grandes pour qu'on ne risque pas de débordement ; de plus elles logent sur un nombre entier de secteurs (3 secteurs de 28 mots, d'où 42 entrées par sous-table).
- on charge la sous-table
- on y fait une recherche séquentielle, d'où le numéro de la transformation
- on se trouve alors ramené au cas 2.

4) accès aux commentaires

On a vu que l'ancienne version prévoyait l'enregistrement des commentaires des transformations dans un fichier supplémentaire. Son utilisation a été rationnalisée de la façon suivante :

- le seul cas ou l'on risque d'avoir besoin de commentaires de transformations semble être le cas où le résultat d'une transformation mérite des éclaircissements. Or, on connaît le nom de la transformation qui a amené à un précurseur lorsqu'on le visualise sur l'écran ; il fallait donc établir une correspondance nom-adresse des commentaires.
- c'est pourquoi on a créé une telle table de correspondance, d'emploi et de structure identique à celle relative aux transformations.

Il reste un cas particulier important à étudier : parmi les 68 caractères, cinq peuvent se subdiviser. Il s'agit de :

EXCHANGES groupe 1 / groupe 2 (échange groupe 1 en groupe 2)
MAKES SUBSTITUTION gra 1 / gra 2 (substitue le groupe 2 au groupe 1)
INTRODUCES groupe
MAKES groupe
REMOVES groupe

Or il existe actuellement 45 groupes fonctionnels, et ce nombre est susceptible de s'accroître avec le développement de la chimie du phosphore. De plus, on peut compter sur le fichier tel qu'il est actuellement, que le caractère EXCHANGES G1/G2 est employé dans 41,6 % des réactions.

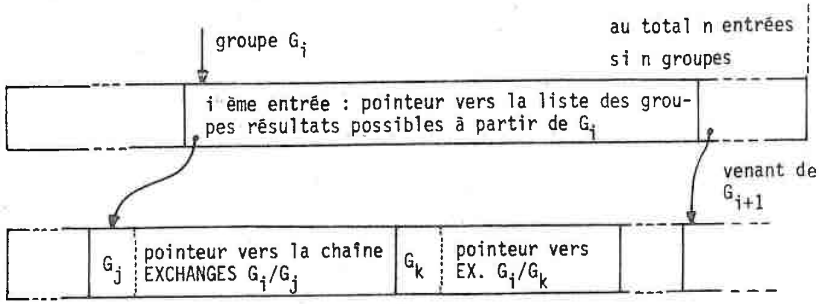
On est donc en présence de deux données contradictoires : si l'on veut réduire l'emploi de EXCHANGES G1/G2, il faut distinguer les cas par couple (G1/G2), mais on augmente considérablement le nombre de chaînes de bits.

En ce qui concerne les quatre autres caractères spéciaux, le problème est du même genre, mais il revêt une importance moindre. On étudiera donc en premier lieu le cas EXCHANGES.

Dans le meilleur des cas, on a 45 groupes fonctionnels ; l'interchange G1 → G1 n'existant pas, et l'opération d'échange n'étant pas commutative, il y aura 45 * 44 = 1980 couples d'interchanges possible. Or une statistique sur le fichier actuel permet d'estimer à environ 300 couples d'interchanges effectivement employés (rapporté à 1 000 transformations), dont seulement 50 sont faisables par plusieurs transformations, avec un maximum de 30 transformations pour un seul interchange. Il en résulte que la méthode consistant à représenter les 1980 couples n'est pas réaliste.

Quoi qu'il en soit, si l'on veut représenter seulement les 300 caractères correspondant aux couples d'interchange existant, on quadruple le nombre de chaînes de bits, et il devient impossible de les traiter toutes en mémoire. Après l'étude (voir PFAADT (21)) on a opté pour la solution suivante :

- on conserve une chaîne de bits globale pour chacun des cinq caractères spéciaux. Seules ces chaînes seront traitées en mémoire
- on définit sur disque les chaînes de bits précisant ces caractères spéciaux, accompagnées de tables d'accès direct en fonction des groupes utilisés (codés par des numéros consécutifs de 1 à 45) :



La même représentation est employée pour le caractère spécial MAKES SUBSTITUTION gra 1/gra 2 (il y a 13 groupes d'atomes concernés). Quant aux trois derniers, qui ne font plus intervenir de couples, on employera le même principe avec un seul niveau pour les tables d'accès.

La planche E représente un schéma récapitulatif du fichier des transformations. On distingue trois fichiers physiques : FICHTABIT, CORAPRAL et FICHCOM.

En conclusion, nous dirons qu'on a obtenu une organisation pratique du fichier des transformations qui permet entre autre de connaître facilement l'ensemble des transformations qui possèdent un ou plusieurs caractères donnés, et ce avec un minimum d'accès au disque.

222 - Nouvelle structure du système

On a d'une part le fichier organisé de façon à permettre un accès rapide aux transformations d'un ensemble donné, et d'autre part la "table de vérité" décrivant l'objectif à atteindre, donc les ensembles à rechercher.

Décrivons sommairement la structure d'une table de vérité (les détails de la représentation seront vus au chapitre III) :

liste du code objet des directives	nombre de directives	nombre de colonnes dans la table de vérité	liste des adresses des directives en fonction de leur rang	colonnes de la table de vérité proprement dite

(voir exemple ci-après)

FICHTABIT :

secteur 0	données générales
1 à 39	table d'adresses des transformations adresse en secteur | longueur en mots 1024 mots
40 à 79	table d'adresses des commentaires et références bibliographiques adresse en secteurs | longueur en mot 1024 mots
80 à 699	table d'accès par les noms des transformations 203 sous-tables de 42 double-mots nom de la transformation (suite) sur 10 caractères | adresse (sect.)
700 à 829	chaîne de bits d'occupation BITOC et chaînes de bits des caractères sous forme d'un tableau de dimension 100 x 32 mots (une chaîne = 32 mots)
830 à 849	chaînes de bits des partitions : PAIR, AROM, SING, PATRN, INTRO, INTER 2 secteurs par chaîne
850 à 899	table d'accès aux chaînes spéciales
à partir du secteur 900	chaînes de bits spéciales, chacune sur 2 secteurs, accessibles par leur numéro de secteur

PLANCHE E(1) : organisation complète du fichier.

FICHCOM :
(fichier des commentaires et références bibliographiques)

Commentaires sur chaque transformation :
- sur un nombre entier de secteurs
- les blancs en fin de ligne sont supprimés

CORAPRAL :

(Cumulative ORganic And Phosphorus Reactions Applied to Logistic)

zone des descripteurs (répertoire)

description sommaire de la sous structure	
adresse de la transf.	type

} 2 mots par transformation

(2048 mots, 74 secteurs)

zone des transformations

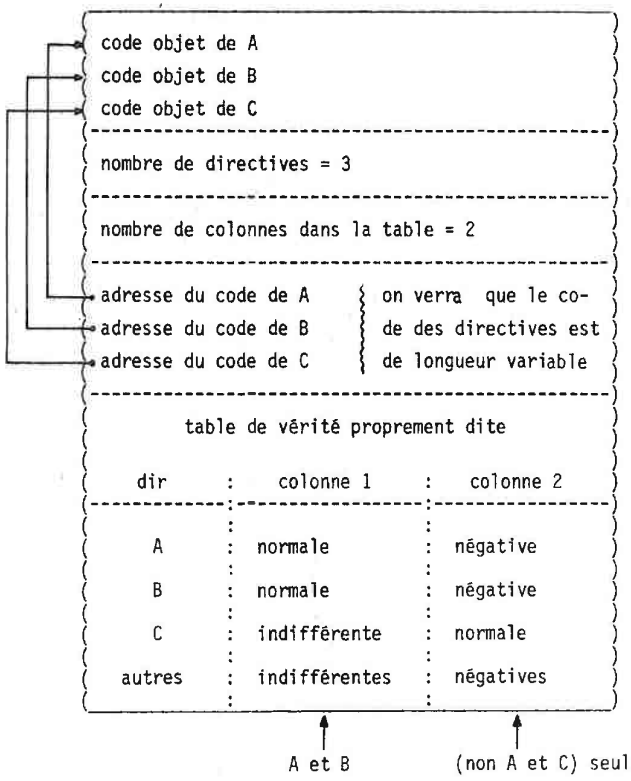
code généré :
- conditions
- enquêtes
- manipulations...

} par transformation

PLANCHE E(2) : organisation complète du fichier (fin).

Exemple : on veut représenter
(A et B) ou ((non A et C) seulement)

on obtiendra :



On se souvient que chaque colonne de la table de vérité est en fait un monôme de la représentation de l'objectif (CLD) sous sa forme disjonctive minimale, et qu'on pourra les étudier l'un après l'autre en concaténant leurs résultats dans l'arbre de synthèse. On va donc commencer par étudier les problèmes concernant l'emploi d'une seule colonne pour atteindre l'objectif donné.

2221 - Problèmes au niveau d'une colonne de la table de vérité

1) directives employées avec négation

Elles devront être traitées à part, sans qu'on puisse employer les chaînes de bits. En effet, la chaîne de bits correspondant à la directive A (éventuellement après combinaison de chaînes-caractères) représente l'ensemble des transformations susceptibles d'exécuter A ; le complément de cette chaîne représente donc l'ensemble des transformations qui ne peuvent certainement pas exécuter A (sauf peut-être par pur hasard). Or on aurait besoin dans ce cas de l'ensemble des transformations qui n'exécutent effectivement pas A.

Exemple : si l'on demande "ne pas casser la liaison entre les atomes 2 et 5", et que ces atomes sont des carbones, on fera appel à la chaîne de bits correspondant au caractère "BREAKS CC BOND". Cette chaîne représente l'ensemble des transformations qui cassent une liaison CC ; mais il peut y en avoir parmi elles qui ne cassent pas celle qui relie l'atome 2 à l'atome 5. Ces transformations ne sont pas dans le complément de la chaîne A, mais elles sont pourtant permises.

De plus, il faut bien voir qu'une directive négative ne demande pas à proprement parler une action.

Il résulte de ceci que l'exécution des directives employées avec négation ne pourra être vérifiée qu'après exécution des transformations. En particulier, une colonne de la table de vérité qui ne contiendrait qu'une seule directive, employée avec négation, nécessite l'essai de toutes les transformations du fichier.

2) passage aux chaînes de bits correspondant aux directives

Ce passage étant bien déterminé et fixe pour chaque directive, il pourra être fait une fois pour toutes avant le début d'exécution de la CLD. A partir de maintenant, lorsque nous parlerons de chaîne de bits dans ce chapitre, il s'agira de celles associées aux directives.

3) différences entre les cas "1 étape"et "n étapes"

Contrairement aux apparences, on ne peut pas considérer le cas où l'objectif doit être exécuté en une étape comme un cas particulier de

celui où on permet n étapes.

Notons tout d'abord qu'on a jugé prudent de limiter n au nombre de directives sans négation employées dans la colonne de table de vérité en cours d'étude (ou 1) ; en effet,si n est supérieur à ce nombre il y aura nécessairement au moins une étape où le système ne progresse pas, tout au moins dans le sens de ce que le sous-objectif demande. Ce cas pourra à nouveau être envisagé plus tard si le système devient capable d'évaluer et de comparer lui-même des chemins en cours de construction ou des molécules, car il lui restera alors quand même un guide.

⊕ Cas "une étape"

- toutes les directives sans négation doivent être exécutées simultanément par chaque transformation retenue : on recherchera donc l'ensemble des transformations qui possèdent à la fois tous les caractères demandés : il faudra faire l'intersection des chaînes de bits correspondantes
- toutes les directives négatives doivent être exécutées dans chaque précurseur pour qu'il soit accepté

⊕ Cas "n étapes"

- toutes les directives sans négation doivent être exécutées au bout des n étapes (ou avant). Dans les n-1 premières étapes, il faudra donc rechercher l'ensemble des transformations susceptibles d'exécuter au moins l'une de ces directives : il faudra faire la réunion des chaînes de bits correspondantes. Au contraire, dans la dernière étape, on se retrouvera dans le cas "une étape" avec les directives qui restent à exécuter.
- il découle du point précédent qu'il faudra conserver au niveau de chaque précurseur obtenu le sous-ensemble de la colonne initiale de table de vérité qui reste à exécuter (ou son complément).
- toutes les directives négatives doivent être exécutées dans tous les précurseurs intermédiaires de chaque étape et non seulement dans ceux qui terminent un chemin.
- à chaque étape, il devra y avoir au moins une nouvelle directive sans négation exécutée. Il ne suffit pas que le nombre de directives sans négation exécutées soit augmenté, sinon on risquerait d'introduire des boucles où le résultat d'une étape à un endroit donné de la molécule serait détruit par la suivante.

Nous voyons donc qu'on trouvera intérêt à isoler ces deux cas et à traiter le cas "n étapes" en décrémentant n à chaque niveau pour terminer à la fin sur "1 étape" avec ce qui reste.

4) Cas où les directives de la colonne doivent être exécutées à l'exclusion de toute autre

Nous l'appellerons pour simplifier "cas ONLY". On a déjà vu que dans ce cas, aucune directive n'est indifférente (sauf après minimalisation éventuelle entre colonnes "ONLY"). On recherchera donc le complément de l'ensemble des transformations qui possèdent l'un au moins des caractères interdits, donc de celles qui ne possèdent que des caractères permis : il est représenté par la "chaîne ONLY" de la colonne. Une fois déterminée la chaîne de bits correspondant aux directives demandées, il faudra en faire l'intersection avec la chaîne ONLY pour obtenir la chaîne de bits représentant l'ensemble des transformations qui ne possèdent que les caractères demandés.

Notons que ce procédé diminue le nombre de transformations à essayer , mais n'exclut pas une vérification a postériori. De plus, on devra employer les chaînes globales correspondant aux caractères décomposables (MAKES groupe, etc...).

Il convient toutefois de remarquer que, dans le cas d'une exécution en n étapes, il faudra faire la construction de la "chaîne ONLY"pour chaque précurseur intermédiaire à partir de l'ensemble des directives qu'il reste à exécuter à son niveau. Ceci nécessite donc soit que la table des chaînes de bits réside en mémoire, soit qu'on la recharge au début de chaque exécution à partir d'un nouveau précurseur.

Or, en l'absence de la chaîne ONLY, on risque d'essayer des transformations impossibles que la chaîne ONLY permettrait d'éviter, ce qui fait que l'accès au disque nécessaire pour recharger la table des chaînes de bits sera dans la majorité des cas compensé par le nombre d'accès évités pour de telles transformations.

5) vérification "objectif atteint"

Toutes les opérations décrites précédemment permettent de fournir l'ensemble des transformations du fichier qui peuvent être essayées. Certaines seront rejetées parce qu'elles ne s'appliquent pas à la molécule étudiée. Pour les autres, il faut vérifier si elles permettent d'atteindre l'objectif (ou simplement de progresser). Plus tôt cette vérification pourra être faite, moins on risquera de perdre du temps.

Or on a vu au chapitre I, que l'interpréteur des transformations du programme de synthèse (SYNTRP) traduit les instructions de manipulation de la transformation en fonction de la numérotation des atomes de la cible courante.

Il y aura un premier niveau de vérification à ce moment : certaines directives sont exécutées par un nombre limité (en général 1) et fixe de telles instructions de manipulations. Par exemple : "rajouter une charge + à l'atome 3" ne peut être exécuté que par une manipulation unique et bien précise. Il suffit donc pour ces directives de regarder si la (les) instruction(s) de manipulation est (sont) présente (s) (ou absentes dans le cas de directives avec négation).

Pour toutes les autres (+ cas ONLY), il faudra construire la table de connexions puis les ensembles d'atomes, de cycles et de groupes fonctionnels attachés au nouveau précurseur, pour pouvoir les vérifier.

En conlusion, nous voyons qu'avant de débuter l'exécution d'une CLD, nous pourrons créer pour chaque directive employée :

- la chaîne de bits correspondante, sauf si la directive est toujours employée avec négation
- éventuellement l'ensemble des instructions de manipulation à rechercher après interprétation des transformations par SYNTRP, ensemble qu'on pourra placer dans la zone des tableaux dynamiques.

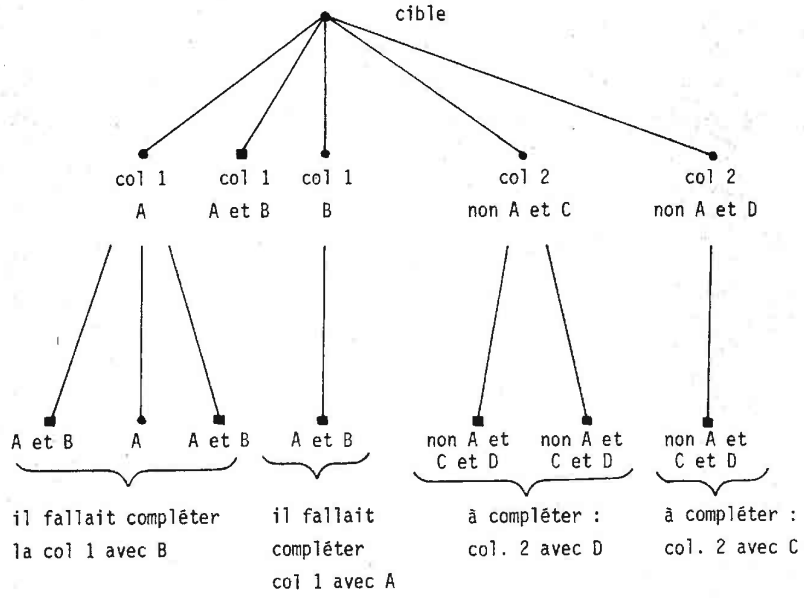
2222 - Utilisation de toutes les colonnes de la table de vérité

* 1) En principe, l'exécution d'une CLD commence toujours sur une cible unique : où qu'on en soit arrivé dans l'arbre de synthèse, on fait le choix du précurseur à partir duquel on veut continuer et qui devient cible.

Sur cette cible, on devra essayer les unes après les autres toutes les colonnes de la table de vérité, en concaténant les résultats obtenus. Mais lorsque l'exécution peut être faite en plusieurs étapes, tous les précurseurs de chaque niveau n'en seront pas nécessairement au même point et n'employeront pas nécessairement la même colonne.

Exemple :

- colonne 1 = A et B
- colonne 2 = non A et C et D
- nombre d'étapes = 2



Il en résulte que :

- il faut conserver dans chaque précurseur intermédiaire le numéro de la colonne commencée ainsi que son état d'avancement
- il y aura une phase d'initialisation pour la cible, sur laquelle on doit essayer toutes les colonnes.

* 2) On a vu dans le 2221 que la recherche des chaînes de bits se fait globalement au début de l'exécution de la CLD et éventuellement à chaque précurseur pour les chaînes ONLY. Elles ne devront être présentes en mémoire qu'à ces moments, et la place qu'elles occupent (jusqu'à 3200 mots) pourra être libérée autrement. On conservera toujours :

- les chaînes types (PATTRN...)
- une chaîne par directive.

Or le reste de l'exécution nécessite la présence en mémoire du répertoire du fichier, qui occupe 2048 mots au maximum. On pourra donc le mettre à la place des chaînes de bits. Il y aura en plus la place d'une sous-table de correspondance nom de transformations-commentaires (42 double-mots).

* 3) Voici donc à partir de tout cela l'algorithme décrivant la partie recherche-balayage du fichier vérifications (en pseudo-algol) :

```
INTCLD : début
charger les chaînes de bits normales ;
pour chaque directive i employée dans la CLD faire
    début
    INTDIR ;                                  ---------- vérifications sur la
    si erreur alors                                      directive; chaîne-dir
        début                                            =CHBIT(i), éventuel-
        message-erreur ;                                 lement instr. manip.
        suppression des colonnes qui emploient cette directive
        fin du cas erreur ;
    fin de la boucle vérifications/ch. bits-dir/instr manip ;
libérer la place des chaînes de bits :
si table de vérité vide alors             ---------- (à la suite d'erreurs)
    début SUCCESS = FALSE ;
        aller en FIN-INTCLD
    fin ;
charger le répertoire à la place ;
FINOBJ = TRUE ;
SUCCESS = FALSE ;
pour chaque colonne C non nulle de la table de vérité faire - boucle d'initiali-
    début                                                    sation sur la cible.
    STEPS(C)= max(1, inf(STEPS donné, nb de dir > 0 de C)); -STEPS=nb d'étapes
    si STEPS(C)= 0 alors aller en FIN-COL ;
    NOC du BS-cible = C ;                     ---------- BS = bloc de struc-
    colonne du BS-cible = colonne C ;                    ture.
    DETCHB (colonne C, STEPS (C)) ;           ---------- détermine chaîne de
                                                         bits de la colonne C:
                                                         RESBIT.
```

```
      si RESBIT vide alors aller en FIN-COL ;
      SUCCESS du BS-cible = FALSE ;                    ┌ traitement de l'obj
      TROBJ (BS-cible) ;                  ---------    │ tif (voir ci-dessou
FIN-COL :                                              └ fait SUCCESS=TRUE
fin de la boucle d'initilisation pour la cible ou fin du travail si STEPS = 1 ;

NEXT-ETAPE :
si FINOBJ alors aller en FIN-INTCLD ;
FINOBJ = TRUE ;
SUCCESS = FALSE ;
pour chaque BS-D faire                          ---------- BS-D= bloc de
      début                                                structure du dernie
      si STEPS dans le BS-D = 0 alors aller en FIN-CIBLE-D ; niveau de l'arbre.
      DETCHB (colonne du BS-D,STEPS du BS-D) ;  ---------- d'où RESBIT de ce
      si RESBIT vide alors                                 qui reste à faire
            début supprimer la filiation vide dans l'arbre ;
                  aller en FIN-CIBLE-D
            fin ;
      SUCCESS du BS-D = FALSE ;
      TROBJ (BS-D) ;                            ---------- étape suivante
      si NOT SUCCESS du BS-D alors supprimer filiation impossible ;
      si NOT SUCCESS alors aller en FIN-INTCLD ;
      FIN-CIBLE-D :
      fin d'une étape supplémentaire sur un précurseur du dernier niveau/boucle ;
aller en NEXT-ETAPE ;
FIN-INTCLD :
fin de INTCLD


PROCEDURE DETCHB (SSOBJ, STEPS) ;               ---------- SSOBJ= reste de
début                                                      la colonne à véri-
si STEPS = 1 alors                                         fier; STEPS= nb
      début                                                d'étapes restantes
      chaîne résultat RESBIT = plein 1 ;
      pour chaque dir positive de SSOBJ faire
            RESBIT=RESBIT AND chaîne CHBIT de cette directive
      fin
```



```
sinon
      début
      chaîne résultat RESBIT = plein 0 ;
      pour chaque dir positive de SSOBJ faire
            RESBIT=RESBIT OR chaîne CHBIT de cette directive
      fin
fin de DETCHB


PROCEDURE TROBJ (BS-D) ;
début
pour chaque réaction retenue dans RESBIT (bit à 1) faire --- en fait, cette bou-
      début                                                    cle est faite par
      si possible d'après la sous-structure alors              type de réaction
            début                                              (PATTRN...) .
            SYNTRP ;                          ---------- interpréation, d'où
            si KILL alors aller en TUE ;                 un nouveau précur-
            vérification sur les instructions de manipulation :  seur NPR
                si STEPS de BS-D = 1 alors
                    si toutes les dir vérifiables sont exécutées alors
                        garder cette transformation
                    sinon la supprimer
                sinon
                    si toutes les dir négatives vérifiables sont exécutées alors
                        garder cette transformation
                    sinon la supprimer ;
            TUE :
            fin de l'essai (SYNTRP) d'une réaction ;
      fin de la boucle sur les réactions à essayer = bits 1 de RESBIT ;
pour chaque réaction retenue faire
      début
      créer sa table de connexions ;
      perception ;                                ---------- d'où les ensembles...
      évaluation ;                                ---------- suppression des dou-
      si possible alors                                        bles instables...
            début
            si STEPS du BS-D = 1 alors
```

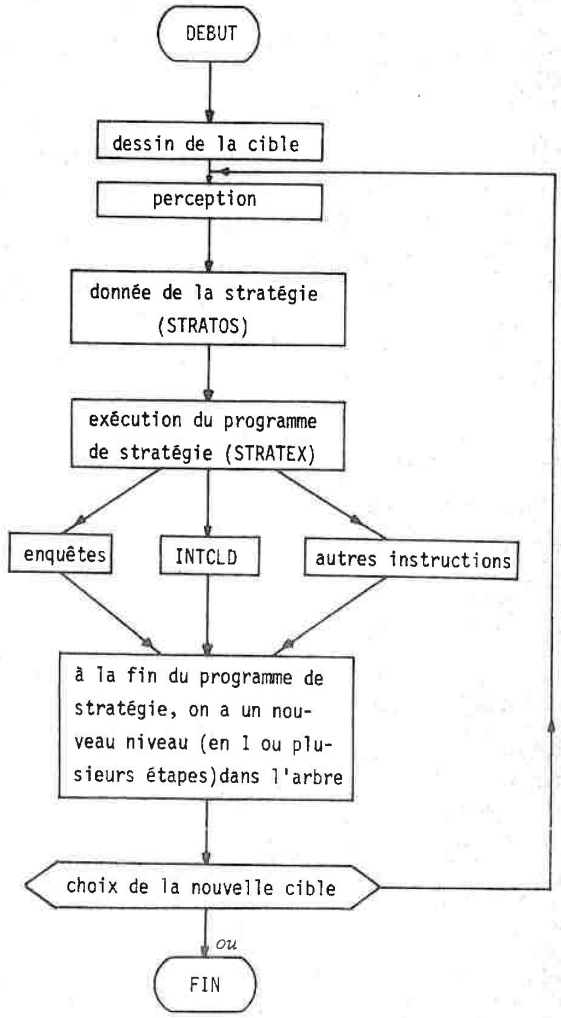
```
            pour chaque dir de la colonne du BS-D faire
                début
                vérification a postériori sur la table de connexions, etc
                si non exécutée alors aller en SUPPRIM
                fin de la dernière vérif "obj atteint"
        sinon début
            pour chaque dir de la colonne du BS-D faire
                début
                vérification a postériori sur la table de connexions, etc
                si dir négative et non exécutée alors aller en SUPPRIM
                fin ;
            si aucune nouvelle dir positive exécutée alors aller en SUPPRIM
            fin ;
        afficher la molécule ;
        demander l'avis du chimiste ;
        si mauvais alors aller en SUPPRIM ;
        chaîner un nouveau BS pour le précurseur obtenu (qu'on appellera
        BS-N).
        STEPS du BS-N = STEPS du BS-D - 1 ;
        SUCCESS du BS-D = TRUE ;
        si STEPS du BS-N = 0 alors FINOBJ = FALSE
        fin de l'étude d'un précurseur possible après évaluation
    sinon
    SUPPRIM : supprimer la réaction et le précurseur
    fin de la boucle sur les réactions retenues
fin de TROBJ.
```

On distingue bien dans cet algorithme les trois phases indépendantes :

- vérifications directives-cible + génération des chaînes de bits-directives
  + génération des instructions manipulation éventuelles
- 1ère étape : essai de toutes les colonnes de la table de vérité sur la cible

- étapes suivantes éventuelles : suite de la colonne commencée pour chaque précurseur intermédiaire.

A partir de là, on peut donner la structure générale du nouveau système avec "stratégie interactive" :



Conclusion :on a mis ainsi à la disposition du chimiste un système qui lui permet de diriger de l'extérieur le développement de l'arbre de synthèse. Ceci permet en particulier de limiter le temps de recherche dans le fichier, et le nombre de précurseurs inutiles. Mais c'est toujours le chimiste qui doit décider de la valeur d'un chemin ou d'un précurseur.

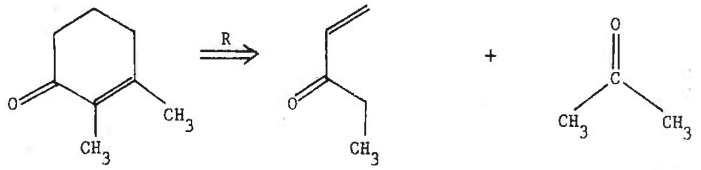
23 - LES AUTOMATES DE STRATEGIE

Jusqu'ici, le système n'a pour "connaissances chimiques" que le fichier des transformations. Or, en synthèse, le chimiste utilise ses connaissances, non seulement pour choisir parmi des réactions possibles, mais aussi parmi des stratégies et des chemins de synthèse qu'il a déjà essayés.

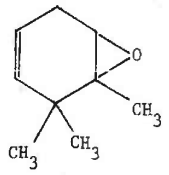
Il est donc naturel, dans une seconde étape, d'essayer de "donner des connaissances" au système dans ce domaine-là.

On pourrait définir un automate de stratégie comme un générateur automatique de chemins de synthèse optimaux utilisés à un cas bien particulier.

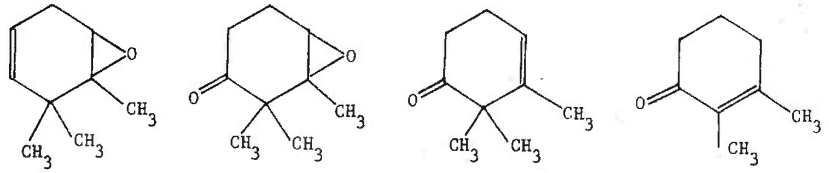
Considérons par exemple la transformation R



qui correspond à la réaction d'annélation de Robinson, particulièrement puissante pour la synthèse de dérivés carbocycliques. Si l'on soumet au programme la cible



la transformation R ne sera pas appelée, car la cible ne contient pas la sous-structure voulue. Par contre, en écrivant un automate spécialisé dans l'application de R, le chimiste pourra prévoir des enquêtes visant à déterminer les différences entre la cible étudiée et la sous-structure nécessaire à l'application de R. Dans chaque cas, il définira les actions à entreprendre pour réduire ces différences ; dans notre exemple : introduire un groupe cétone, puis une double liaison, et enfin amener la double liaison en alpha du carbonyle, suivant la séquence



I II III

On pourra bien appliquer la transformation R sur le précurseur intermédiaire III; l'automate a ainsi créé directement un chemin de quatre arcs.

On constate donc que si une transformation permettait, à partir d'un certain type de molécules, de fabriquer un arc dans le graphe, un automate permettrait de créer un (ou quelques) chemin(s) complet(s), ce qui est possible par associativité de la concaténation des arcs. On s'oriente donc vers une recherche de l'arbre en profondeur d'abord.

Il apparaît d'emblée que cette méthode peut donner lieu à diverses extensions :

- le chimiste définit les automates et les appelle lui-même au moment voulu dans son programme de stratégie interactive
- l'appel des automates est automatique
- la définition de nouveaux automates se fait automatiquement à partir des programmes de stratégie interactive donnés par les utilisateurs.

Nous allons étudier successivement ces trois cas.

231 - Définition et appels d'automates

Notons tout de suite que les règles que nous allons définir ici seront assez générales sur les deux autres cas : il s'agira plutôt d'automatiser leur application.

Le principe de l'automate de stratégie est le suivant :

Soit E l'ensemble des molécules possédant une propriété P donnée pour laquelle on sait définir, pour certains éléments de E,un chemin de synthèse intéressant classique ; il peut y avoir certains éléments pour lesquels on ne peut pas conclure de façon générale. La construction du chemin est basée soit sur le fait qu'on désire passer par un arc bien précis, soit par un ensemble de points bien précis. L'évaluation d'un chemin peut être faite par l'utilisateur ou éventuellement par l'automate si l'on sait définir suffisamment de critères d'évaluation à l'écriture de l'automate.

Cette définition est évidemment très générale ; dans l'état actuel des choses, on envisage principalement certains cas :

- la propriété P est le fait de contenir un motif structural chimique donné ; l'ensemble E est alors l'ensemble des molécules qui renferment ce motif. Notons que cet ensemble ne peut être défini en extension, mais que sa fonction inclusion est parfaitement définie.

- définition de chemins de synthèse intéressants qui passent par un arc ou un ensemble de points donné : on sait par expérience qu'une certaine réaction est particulièrement adaptée à la synthèse de composés renfermant un motif donné (donc de l'ensemble E). Mais la transformation correspondante ne peut pas nécessairement être appliquée à toute molécule de l'ensemble E, car la présence de tel autre motif, ou groupe... peut empêcher le fonctionnement ou tout au moins le rendre par trop peu rentable. Il faudra donc éventuellement "préparer le terrain" avant de pouvoir lancer l'exécution de la transformation en question. On définit bien ainsi un chemin qui permet de faire passer par un arc donné (la transformation).

- évaluation faite par l'automate : il est possible que dans certains cas, après l'exécution d'une étape supplémentaire dans la construction d'un chemin, on puisse tester dans l'automate la structure du précurseur obtenu et qu'on puisse en déduire si le chemin peut être continué ou non. Dans d'autres cas, il faudra avoir recours à l'appréciation du chimiste, ou à des méthodes plus sophistiquées.

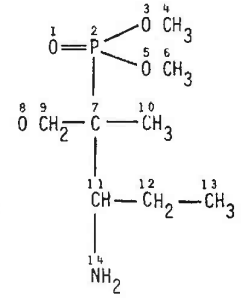
On trouvera en annexe B l'organigramme de l'automate basé sur l'emploi de la réaction d'Arbusov défini par M.H. ZIMMER pour définir des chemins de synthèse pour des composés contenant le motif  .

On déduit de tout ceci qu'un automate se comporte comme un programme de stratégie préétabli pour pouvoir servir dans un certain nombre

de cas. C'est donc en quelques sortes un sous-programme de la stratégie interactive en cours, que l'utilisateur appelle lorsqu'il lui semble qu'elle est applicable à l'un des précurseurs où il vient d'arriver. Le passage des paramètres se fait par la description du motif sur lequel sont définies toutes les phrases de l'automate ; il s'agit en fait d'une correspondance entre les numérotations comme le montre l'exemple ci-dessous :

dans l'automate

dans la molécule étudiée



soit la phrase (dans l'automate) :
IF ATOM ALPHA TO ATOM 3 IS OXYGEN THEN

elle sera interprêtée comme IF ATOM ALPHA TO ATOM 7 IS OXYGEN THEN.

On peut également considérer un automate comme une "macro-réaction", bâtie sur une ou plusieurs transformations, sur des enquêtes, des manipulations, etc... On a pourtant voulu séparer ces deux problèmes ; en effet, un automate demande une manipulation et il faut chercher des transformations susceptibles de la faire ; au contraire, une transformation fait directement les manipulations qu'elle contient. De plus, les automates décrivent réellement une stratégie, une méthode d'utilisation de réactions chimiques,alors que la transformation est une étape élémentaire chimiquement bien définie.

Puisqu'on peut ainsi rapprocher l'écriture des automates de l'écriture d'une stratégie interactive, étudions les particularités de ces deux cas.

* 1) il faut pouvoir réaliser des enquêtes structurales sur l'environnement du motif dans la molécule sur laquelle s'exécutera l'automate.

Ceci n'est évidemment pas nécessaire en stratégie interactive, puisque l'utilisateur voit la cible sur l'écran et définit ses directives en fonction de cela.

* 2) l'exemple d'ALCHEM montre que la définition d'enquêtes structurales peut être grandement simplifiée par l'emploi d'instructions de définition et de manipulation d'ensembles (d'atomes...)

* 3) si l'automate veut faire des calculs (de rendements, etc...) il faudra prévoir des instructions arithmétiques, ce qui serait d'un emploi limité en matière de stratégie interactive puisque là encore on peut afficher ces indications à chaque étape et décider "manuellement" en fonction de cela.

* 4) on doit pouvoir définir le motif (ou autre caractéristique de sélection éventuellement) sur lequel porte l'automate.

* 5) on devra disposer d'instructions permettant de définir des choix : à un moment donné, il peut y avoir plusieurs chemins a priori possibles, pour lesquels on ne peut pas décider sans essayer. Il faudra donc indiquer le point de départ et d'arrivée des descriptions de ces chemins parallèles, de même que leur point de départ dans l'arbre de synthèse. Notons qu'il s'agit d'une notion analogue mais plus générale que les CLD où les OR permettent la définition de choix.

* 6) au contraire, on employera peu (ou pas) de CLD dans les automates, car les actions à entreprendre seront très précises à chaque instant, et souvent réduites à l'appel d'une transformation.

Il en résulte que le langage de définition d'automates aura une partie commune avec celui servant à la stratégie interactive. On verra dans les chapitres III et IV comment ce problème a été résolu.

Etudions maintenant la question de l'appel des automates. Dans le cas où nous nous sommes placés pour l'instant, elle n'est pas très compliquée à résoudre, puisque c'est l'utilisateur qui le demande. Or, par définition de la stratégie interactive, c'est après la réalisation d'un objectif qu'il fera un tel appel : parmi les précurseurs ayant atteint l'objectif précédent, l'un (au moins) est susceptible d' être concerné par un automate ; on le prendra donc pour nouvelle cible courante, et on appellera l'automate en guise de programme de stratégie.

Pour permettre cela, on va constituer une bibliothèque d'automates comme on a une bibliothèque de transformations. Chaque automate aura :

- un nom par lequel il sera appelé ; le nombre d'automates qu'on définira,qui dépend avant tout des recherches des chimistes, dictera la méthode à employer pour gérer la table des noms.

- un critère de sélection, actuellement le motif ; or la recherche d'un motif dans une molécule est un problème ayant déjà une solution puisque les transformations sont également sélectionnées par sous-structure (on aura donc à définir les motifs d'automates comme les PATTRN's).

On voit donc que dans le cas où l'appel est fait par le chimiste, on n'aura besoin que de techniques déjà connues.

Un dernier point mérite d'être soulevé, il concerne l'exécution effective de l'automate.

Une fois l'automate appelé et la correspondance motif-cible faite, c'est un programme de stratégie qui s'exécute, au même titre qu'un autre. Mais ce programme n'a pas été nécessairement écrit par la personne qui l'utilise, et certainement pas juste avant que ne démarre son exécution. Il pourra donc être particulièrement utile de disposer d'un moyen permettant à l'utilisateur de suivre le déroulement de l'automate. On définira donc dans le système une option de trace dont le but est de fournir en clair les instructions de l'automate en cours d'exécution.

232 - appels automatiques

On pourrait, dans un deuxième temps, laisser au système la possibilité de décider lui-même de réaliser l'appel d'un automate. Notons qu'il ne s'agit plus là que de propositions de développements futurs du système. En effet, il vaut mieux d'abord le tester avec les améliorations décrites jusqu'ici avant de trop pousser les études dans d'autres directions.

Dans l'état actuel des choses, on n'envisage que des automates caractérisés par le motif sur lequel ils s'appliquent.Pour réaliser l'appel automatique de tels automates, il faudrait donc que le système essaye tous les motifs d'automates qu'il connaît sur chaque précurseur, y compris sur les précurseurs intermédiaires des stratégies en plusieurs étapes.

Remarquons qu'il s'agit là d'un problème qui s'apparente à une reconnaissance de formes : il faut déterminer rapidement la classe d'équivalence à laquelle appartient une "forme" donnée, qui est ici la cible

étudiée, par rapport à des références constituées par les motifs des automates. Il conviendra donc d'étudier les techniques qui existent dans ce domaine (23).

On peut faire d'emblée les remarques suivantes :

* 1) l'exécution d'un automate devient un véritable sous-programme du programme de stratégie en cours. On sera donc amené à empiler certaines variables qui régissent l'exécution, en particulier celle des CLD, entre autres la table de vérité elle-même.

* 2) il faudra posséder un moyen de dépistage très rapide de la présence des motifs à l'intérieur d'une molécule. On peut envisager d'essayer de faire une sorte de classification des motifs. On trouvera probablement avantage à développer d'abord l'étude de procédés de comparaison de molécules (§ 24), car des résultats obtenus à ce sujet pourront certainement servir à cette fin. De toutes façons, il faudra faire une optimisation très poussée du module de recherche d'une sous-structure dans une molécule (qui existe déjà).

233 - définitions automatiques

Il s'agit d'un problème complexe d'auto-apprentissage : il faut que le système conserve des stratégies données par l'utilisateur ainsi que les motifs sur lesquels elles s'appliquent. Là encore, on pourra s'appuyer sur des résultats généraux déjà obtenus par la recherche dans le domaine de l'auto-apprentissage.

On peut concevoir d'emblée deux problèmes principaux :

* 1) quels critères permettent de décider qu'un programme de stratégie vaut la peine d'être conservé ?

* 2) comment généraliser la stratégie employée sur un motif à tous les cas d'entourage de ce motif, ou de conditions expérimentales ?

Il résulte de ces deux interrogations qu'on trouvera intérêt à réaliser une classification des programmes de stratégie à partir des sous-structures. Le problème devient ainsi tout d'abord une étude des sous-structures rencontrées avec leur environnement, avec en particulier une comparaison des motifs. On aura donc intérêt à développer en premier lieu des méthodes de comparaison, de classification et d'évaluation des molécules.

En second lieu, il faudra être à même d'évaluer la valeur d'un chemin essayé en stratégie interactive, afin de permettre au système de décider de conserver une stratégie ou non.

Le système s'alimenterait donc à partir d'un fichier organisé sous forme d'une pyramide renversée où seraient notées principalement les sous-structures utilisées, leur fréquence d'utilisation (les moins utilisées étant dans la pointe de la pyramide, et vice versa) et les liens avec les "plus proches" et avec les programmes de stratégie correspondants. Lorsqu'une sous-structure atteint un certain niveau dans la pyramide, le système regroupe ses programmes de stratégie pour en faire un automate cohérent qui sera alors inséré dans la bibliothèque des automates.

234 - En conclusion de ce chapitre sur les automates, nous voyons que si la constitution d'automates ne pose pratiquement plus que le problème de leur écriture par les chimistes, la question de leur appel automatique et surtout de leur constitution automatique relève encore pour une bonne part du rêve. En particulier, il est important d'avoir développé auparavant des méthodes de comparaison et d'évaluation de molécules et de chemins.

24 - COMPARAISON DE MOLECULES ET DE CHEMINS

C'est le point dont l'étude semble actuellement prioritaire ; une action vient d'être lancée à ce niveau. Nous allons étudier brièvement le problème tel qu'il se présente, mais il n'est pas encore possible de conclure, ni même de le développer.

241 - Comparaison de molécules

Deux cas sont à considérer : évaluer la valeur, l'intérêt d'une molécule, ou comparer des molécules entre elles et définir la "distance" qui les sépare.

Dans le premier cas, on sera amené à définir une "fonction d'évaluation" travaillant sur la table de connexions et sur les ensembles qui s'y rattachent. Rappelons qu'il existe déjà un module d'évaluation dont les rôles principaux sont de supprimer des doubles dans l'arbre ou des précurseurs chimiquement instables... Il faudrait donc perfectionner ce module pour ne plus se limiter à l'étude d'impossibilités, mais pour être en mesure

de donner une "note" à chaque molécule rencontrée. On pourrait ainsi introduire en stratégie des contraintes du type :

- ne garder que les n précurseurs les plus importants
- ne garder que ceux dont la note est supérieure ou égale à n,

et caetera.

Une telle fonction d'évaluation serait basée par exemple sur des critères tels que nombre de cycles, présence de groupes particuliers... L'un des premiers problèmes à résoudre sera sans doute de trouver des critères qui ne changent pas avec le type de problème étudié.

Dans le second cas, il ne suffira plus d'évaluer les précurseu dans l'absolu, mais relativement à d'autres molécules. Les nouvelles possibilités qu'on pourrait espérer donner de cette façon au système seraient par exemple :

- essayer de se rapprocher et d'atteindre une molécule connue qu'on pense a priori pouvoir être un précurseur possible
- sélectionner dans un fichier de produits commercialisés ceux qui semblent être "les plus proches "de la cible, puis essayer de les atteindre
- en définissant la "distance" comme un nombre relatif, on pourrait avoir un "sens positif" correspondant à un déplacement dans le sens de la simplification des précurseurs, et un "sens négatif" impliqué par le déplacement vers des molécules plus complexes. L'apparition d'une distance négative impliquerait l'arrêt de la stratégie en cours.

Là encore, les critères utilisés pour la définition de ces distances ne doivent pas varier d'un cas à l'autre, et c'est peut-être là que se situera la principale difficulté à surmonter.

Une solution semble pouvoir être envisagée pour commencer la recherche dans ce domaine. Il apparaît que c'est le plus souvent d'après le dessin de molécules que le chimiste fait des observations concernant les similitudes entre structures. L'idée consiste donc à essayer d'adapter une méthode de représentation de dessins par des grammaires telles que celles que proposent R.S. LEDLEY et J.B. WILSON (24). De plus, le système PASCOP utilise déjà l'algorithme de Morgan (12,25) pour donner à chaque structure une représentation équivalente indépendante de la façon dont en a été fait le dessin (symétries, ordre d'entrée des atomes...). On peut espérer que la conjonction de ces deux méthodes permet d'obtenir une représentation des structures

sur lesquelles les comparaisons et les recherches de différences élémentaires soient simples.

242 - Comparaison de chemins de synthèse

On débouche ici sur des problèmes très pratiques (mais la synthèse n'est-elle pas avant tout une opération pratique ?) : il s'agit du coût, du rendement d'un chemin, ou d'autres notions de ce genre.

Pour cela,on associera à chaque arc, donc à chaque transformation, une fonction de coût telle que si x_0 à x_i sont des points déjà construits sur le graphe, et x_{i+1} le dernier juste atteint, alors on a $f(x_0 \ldots x_{i+1}) = f(x_0 \ldots x_i) \oplus f(x_{i+1})$, où $\oplus$ désigne une opération associative telle que l'addition ou la multiplication et $f(x_j)$ est la fonction représentant le coût de l'arc reliant x_{j-1} à x_j. De plus, cette fonction de coût et l'opération $\oplus$ doivent être telles que si $(x_0 \ldots x_i)$ est le chemin de i arcs dont la fonction de coût $f(x_0 \ldots x_i)$ est minimale, et si (x_i, x_{i+1}) est l'arc d'origine x_i dont la fonction de coût est minimale, alors $x_0 \ldots x_{i+1}$ doit être le chemin de i+1 arcs dont la fonction de coût $f(x_0 \ldots x_{i+1})$ est également minimale. Il pourrait s'agir par exemple de somme des prix de revient de produits utilisés, ou de produits de rendements de réactions exprimés en pourcentage. De plus, ce problème sera certainement lié à celui de l'évaluation des molécules obtenues après chaque transformation.

Il serait ainsi possible :

- de classer des chemins l'un par rapport à l'autre
- de stopper la construction d'un chemin dont le rendement devient trop faible (ou le prix trop élevé, etc...).

Rappelons qu'il existe déjà dans les transformations la notion de PRIORITY, qui correspond à une évaluation (très subjective) de la transformation. On sera donc amené à préciser cette notion. De plus, on pourra se rapprocher d'algorithmes classiques de la théorie des graphes : recherche de coût minimal, du flot maximal, etc...(16)

En conclusion, on voit que, ce cas faisant appel à des notions plus pratiques, il sera peut être plus simple de définir les besoins et les objets à créer pour les traiter. Il sera toutefois lié au précédent (§ 241).

25 - CONCLUSION

Il ressort de ce chapitre deux éléments principaux. Tout d'abo le choix de la stratégie interactive comme première phase de développement aura été doublement judicieux : il correspond à une première amélioration logique du système du point de vue de l'utilisateur non-informaticien, mais on pourra ainsi s'appuyer sur la structure nouvelle donnée au système pour toutes les améliorations prévues pour la suite.

En second lieu, on voit que si les possibilités du système son largement augmentées par la stratégie interactive et les automates définis par l'utilisateur, le problème est loin d'être fermé, et il donnera lieu encore à de nombreuses recherches informatiques dans le domaine de l'intelligence artificielle.

CHAPITRE III

COMMUNICATION CHIMISTE-SYSTEME AU COURS DE LA SYNTHESE

Comme on l'a vu dans le chapitre II, l'utilisateur sera amené à de nombreux moments à communiquer avec le système :

- dans le cas d'une "stratégie interactive", il devra énoncer les directives qu'il donne au système pour guider ses recherches
- il devra définir ses "automates"
- il lui faudra préciser ses limites, ses contraintes de rendement... pour que le système soit en mesure de faire des comparaisons de molécules, de chemins, etc...

On se rend compte de prime abord que les besoins ne seront pas les mêmes dans tous les cas, mais qu'un certain nombre d'entre eux seront néanmoins communs. En particulier, il en est deux qui sont évidents : d'une part, on veut exprimer des opérations chimiques, et d'autre part, c'est un non informaticien qui le fera. Le langage ALCHEM répondant déjà plus ou moins bien à ces deux conditions, il est clair qu'il faudra en tenir compte dans la définition de ces moyens de communication.

Etudions maintenant les contraintes particulières à respecter dans les principaux cas : stratégie interactive et stratégie par automates.

Comme son nom l'indique, une "stratégie interactive" devra pouvoir être définie en conversationnel, le chimiste travaillant étape par étape, en fonction des résultats qui se construisent sous ses yeux. De plus, le chimiste pouvant voir la structure de la molécule sur laquelle il va travailler, il peut définir directement l'objectif qu'il veut atteindre à l'aide

d'une combinaison logique de directives, sans avoir à faire d'enquêtes préalables. Ce qu'il aura à exprimer sera donc presque exclusivement des opérations chimiques à réaliser sur la molécule en cours d'étude. Ces directives devront pouvoir être reliées entre elles par les opérateurs logiques classiques (et, ou exclusif, ou inclusif), ou être employées avec négation, de façon qu'il soit possible d'exprimer des souhaits du type :

"casser la liaison entre les atomes 2 et 3,
modifier la coordinence de l'atome 5,
ne pas modifier celle de l'atome 1,
ou :
créer un groupe alcool à partir des atomes 2, 3 et 4".

Dans le cas d'un "automate de stratégie", seule la présence d'un motif particulier est connue ; il faut donc prévoir tous les cas qui peuvent se présenter en fonction de l'environnement possible de ce motif. Il devra y avoir en particulier un moyen de faire toutes les enquêtes nécessaires et de commander des choix en fonction du résultat de ces enquêtes. Au contraire, l'emploi des directives demandant des modifications sur la molécule sera beaucoup plus restreint. En effet, on a vu que très souvent les automates seront bâtis autour d'une transformation particulièrement bien adaptée au traitement du motif retenu. L'automate servira donc principalement à "préparer le terrain" pour que la transformation en question devienne applicable.

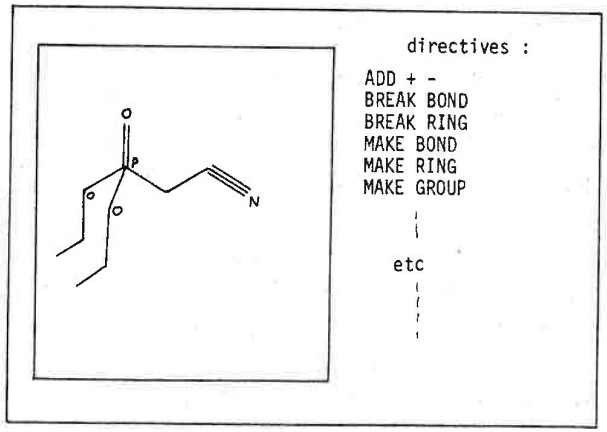
De plus, un tel automate se comportant un peu comme un sous-programme de bibliothèque, il ne sera pas nécessaire de le définir en conversationnel. Il pourra même être judicieux d'autoriser cette opération en traitement batch ordinaire, pour des raisons évidentes de coût.

Il résulte de cette étude que le moyen de communication adopté devra, pour pouvoir recouvrir tous les cas, répondre aux contraintes supplémentaires suivantes :

- permettre de définir les modifications désirées sur la molécule
- permettre de faire des enquêtes
- permettre toute autre instruction n'influençant pas directement la molécule
- pouvoir communiquer avec le système en conversationnel
- garder la possibilité de travailler en batch.

La contrainte impliquée par l'existence du langage ALCHEM va largement limiter les choix. En effet, il faudra définir un "langage " (au sens large) qui soit syntaxiquement aussi proche que possible d'ALCHEM, ou au contraire d'une conception aussi différente que possible, afin d'éviter les risques de confusion entre l'un et l'autre. Dans tous les cas, on essayera de rester très souple quant à la syntaxe et au vocabulaire employé. De plus, il serait souhaitable que l'on puisse corriger très facilement un grand nombre d'erreurs... ou même éviter le plus possible les risques d'erreur.

Une première solution semblait réalisable : la molécule est tracée sur une moitié de l'écran, une liste des actions possibles étant écrite sur l'autre côté :



On sélectionnerait la directive choisie à l'aide du crayon lumineux, puis les éléments de la molécule à traiter par cette directive directement sur le dessin de la molécule.

Si cette méthode est séduisante parce qu'elle permet de travailler directement sur le dessin de la cible en cours d'étude, sans avoir recours à la moindre codification, et sans risque de faute de frappe ou d'orthographe puisqu'il n'y a pas de texte à taper au télétype, on se rend compte tout de suite qu'elle comporte de très nombreux inconvénients :

- les interactions avec le crayon lumineux ne sont pas toujours bien performantes; elles dépendent grandement de la charge de l'Univac.

- pour les directives qui appellent une liste d'atomes ou de liaisons, etc... il faudra en fait autant d'interactions avec le crayon lumineux.

- il n'est guère aisé de définir ainsi une combinaison logique de directives.

- enfin (et surtout), cette méthode est mal adaptée à la construction d'automates, qui s'appuient non sur une molécule complète, mais seulement sur le motif chimique associé et qui utilisent fréquemment des enquêtes dont les opérandes ne peuvent pas toujours être pointés sur l'écran.., surtout s'ils n'y sont pas (exemple : "si l'atome en alpha de... est un oxygène" : cet atome n'est pas sur l'écran). De plus, il est évident que ce système n'est pas utilisable en batch.

Devant l'échec évident de cette méthode de communication, pourtant attrayante au départ, il a semblé qu'il valait mieux se tourner vers un "langage plus classique", dans lequel on définit tout par des "phrases". Dans ce cas, on devra presque obligatoirement rester aussi près que possible d'ALCHEM, tout au moins en ce qui concerne la syntaxe.

Il restait le problème de la compilation de ce langage. Plusieurs méthodes semblent possibles vu le type de langage choisi ; les deux qui semblent les plus dignes d'intérêt sont les suivantes :

- chaque directive et instruction ayant une utilisation bien précise, dépendant de peu de paramètres, on pourrait générer pour chacune un appel à un sous-programme la traitant avec pour paramètres ceux de la phrase source. Par exemple, la directive

BREAK RING 2, 3, 4, 5, 6

serait remplacée par un appel du type

CALL BREAK (RING, 5, 2, 3, 4, 5, 6)

où le premier 5 représente la longueur de la liste qui suit.

- on transforme les phrases en un code objet où tous les paramètres sont compactés dans des champs bien définis.

Les deux méthodes ont paru assez séduisante de prime abord.

Nous avons toutefois retenu assez rapidement la seconde pour diverses raisons :

* 1) la première ne permet pas un gain de place effectif entre la source et le code généré, ce que la seconde permet de façon satisfaisante.

* 2) la plupart des instructions, et les enquêtes, seront reprises telles quelles d'ALCHEM ; il serait donc pratique de pouvoir également reprendre les décodeurs de code ALCHEM correspondants, qui ont fait leurs preuves. Il faudrait donc générer un code objet "compatible ALCHEM".

* 3) on verra dans le chapitre sur les combinaisons logiques de directives (§ 334) que l'emploi d'appels de sous-programmes ne serait guère pratique à utiliser dans ces cas.

C'est ce qui a amené à la constitution du langage STRATOS et de son compilateur, que nous allons maintenant étudier. Le chapitre IV présentera en détail STRATOS vu par l'utilisateur non informaticien. On trouvera un manuel ALCHEM dans (13 - 12).

31 - GENERALITES - DEFINITIONS

Il est apparu immédiatement que certaines notions existant dans ALCHEM pourront être reprises sans changement dans leur syntaxe, ce qui simplifie d'autant l'apprentissage du langage. Il s'agit des enquêtes, des instructions définissant et maripulant des ensembles (d'atomes...), et de celles qui permettent les calculs arithmétiques (priorités...). On remarquera que ces instructions serviront principalement dans les automates, alors que les directives d'accès à la molécule, qui composent les programmes de stratégie interactive, sont toutes à définir. Il sera naturel de les présenter sous forme de phrases chimiques anglaises, de façon à rester conforme avec le reste.

Donnons quelques définitions. On appellera "programme de stratégie", ou simplement "stratégie", l'ensemble des "phrases" que donne le chimiste pour atteindre l'objectif qu'il s'est fixé. Il existe trois catégories de phrases :

- les enquêtes, dont le résultat est "vrai" ou "faux "
- les directives, qui définissent directement ou indirectement une modification que l'on veut apporter (ou éviter) à la molécule
- les instructions, qui permettent toute autre opération dont le résultat n'affecte pas la molécule (arithmétique, ensembles...).

Les directives peuvent être groupées entre elles à l'aide d'opérateurs logiques ; on appellera une telle combinaison logique de directives une CLD.

Les phrases seront composées d'une suite de "mots", un mot étant un nombre, un caractère spécial ou un nom (par exemple, ATOM). Un nom sera défini comme une suite de lettres ou de chiffres, le premier étant une lettre, seuls les 12 premiers étant pris en considération.

32 - PRINCIPAUX CHOIX EFFECTUES

En fonction de ce qui a été dit au début du chapitre, on a été ainsi amené à faire un certain nombre de choix :

- un programme de stratégie doit pouvoir être compilé aussi bien en conversationnel qu'en batch
- la syntaxe et le vocabulaire doivent être aussi souples que possible afin de simplifier l'apprentissage du langage et de limiter les risques d'erreurs
- le système de corrections des erreurs sera particulièrement développé pour utiliser au maximum les ressources du conversationnel
- aussi bien le texte source que le code généré d'une stratégie doivent pouvoir être conservés en fichier pour être réutilisés par la suite, notamment dans le cas des automates.

Nous allons maintenant étudier les méthodes employées pour tenir compte de ces choix.

33 - METHODES DE COMPILATION - STRUCTURE DU COMPILATEUR

STRATOS empruntant à ALCHEM les enquêtes, la structure générale du langage sera,comme ALCHEM, analogue à celle d'Algol :


```
⟨
IF enquêtes THEN
      BEGIN
        ⟨
      DONE
ELSE
      BEGIN
        ⟨
      DONE
⟩
```

Par contre, il n'apparaît nulle part la notion de déclaration au sens d'algol.

De plus, d'un point de vue formel, une CLD se comporte comme une expression booléenne parenthésée classique, où les variables logiques seraient les directives.

Au contraire, on constate que chaque directive et instruction a sa syntaxe bien particulière (voir chapitre IV), et, même s'il existe parfois certaines similitudes, on sera amené à réaliser à part la compilation de chacune de ces phrases (par exemple une place par directive dans une matrice de précédence, ou un sous-programme par directive...).

A partir de ces constatations, on a choisi la méthode suivante :

- définir le langage à l'aide d'une grammaire d'automates (29) dans laquelle directives et instructions sont considérées comme un seul mot (qu'on appellera VERBE-DIR et VERBE-INSTR respectivement). La compilation des enquêtes, à structure de blocs imbriqués, imposera l'emploi d'une pile d'analyse syntaxique (voir § 331).
- analyser les directives et les instructions à l'aide de sous-programmes séparés qui les traitent mot à mot (voir § 335).

Notons que ce mélange de deux méthodes de compilation existe dans des compilateurs, comme cobol et fortran IV H d'IBM (voir GRIES(20)).

331 - Analyseur syntaxique - grammaire

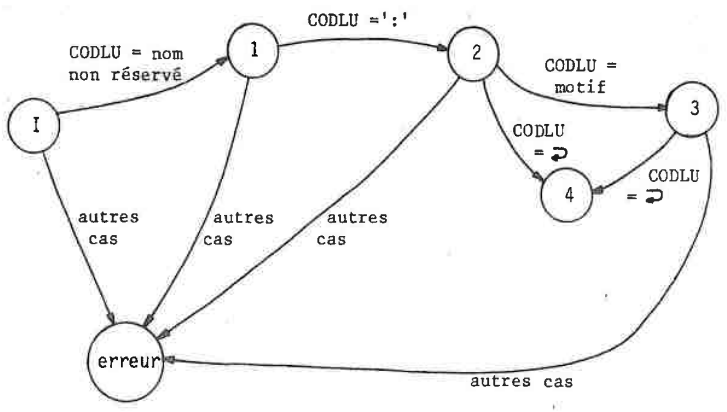
3311 - Principe

La grammaire doit pouvoir représenter en mémoire le langage défini sous forme d'un automate déterministe. Il existe un seul état initial

et un seul état final. Le passage d'un état à un autre est commandé par la nature du nouveau mot envoyé par l'analyseur lexical, ou tout au moins de son code CODLU. Il existe un état particulier, appelé "état erreur", qu'on peut atteindre à partir de chaque état de l'automate. On passe dans l'état erreur chaque fois que le CODLU reçu ne permet de définir aucun autre changement d'état. Pour pouvoir sortir de l'état erreur, on conserve le numéro de l'état où l'on se trouve à des moments bien particuliers de l'analyse (en pratique : en début de ligne).

Exemple : un programme de stratégie doit commencer par un nom (pour le nommer), suivi du caractère ':' , suivi soit d'un motif (dans le cas d'un automate de stratégie) ou de la fin de la ligne (notée ↵ ; dans le cas d'une stratégie interactive) (voir chapitre IV).

La portion correspondante de l'automate peut s'écrire :



L'état (4) est l'état final de cette partie de l'automate.

De plus, on constate en observant la syntaxe du langage (voir chapitre IV) que certaines parties de l'automate se retrouveront sans changement à plusieurs endroits différents. Pour éviter ces répétitions inutiles, on définit des sous-automates.

Un sous-automate est une portion de l'automate ayant un état initial et un ou plusieurs états finals, dans lequel on peut entrer à partir de plusieurs états de l'automate (ou d'un autre sous-automate). L'état dans

lequel on devra passer à la sortie du sous-automate est indiqué au niveau de l'appel. Comme il doit être possible de faire des appels en cascade, il sera nécessaire d'empiler les numéros d'états de retour.

Exemple (simplifié) : une possibilité d'écriture pour les enquêtes est la suivante :

IF <ensemble 1> IS <ensemble 2> THEN

avec

<ensemble 1> :: = <prémodificateur> <opérande 1> <postmodificateur>|
<opérande 1> <postmodificateur>|
<prémodificateur> <opérande 1>|
<opérande 1>

<ensemble 2> :: = <prémodificateur> <opérande 2> <postmodificateur>|
<opérande 2> <postmodificateur>|
<prémodificateur> <opérande 2>|
<opérande 2>

<opérande 2> :: = <opérande 1> | <autres op spéciaux>

On voit donc que <ensemble 1> et <ensemble 2> ont la même structure, mais que <opérande 2> a simplement des valeurs que <opérande 1> ne peut pas avoir.

On définira donc trois sous-automates : <prémodificateur> , <opérande 1> et <postmodificateur>. La portion d'automate correspondante est représentée sur la planche F .

Dans cet exemple, les états 22 et 26 sont les états de retour du sous-automate <opérande 1> (etc...)

De plus, cet exemple montre qu'on sera amené à définir un CODLU particulier qui correspond à l'indication "autres" (toutes les autres valeurs de CODLU), inscrit par exemple au départ de (20) . On notera ce CODLU spécial "Δ".

Un problème que l'on rencontrera relativement souvent sera l'emploi de parties de phrases en option qui peuvent être décrites par sous-automates, telles que <prémodificateur> ou <postmodificateur> de l'exemple précédent. Ce cas sera résolu à l'aide du Δ : lorsqu'un sous-automate décrit une partie optionnelle, il comportera une branche correspondant à "tous les autres CODLU", donc Δ, qui relie son état initial directement à son état final. Si au contraire, la partie décrite par le sous-automate est obligatoire,

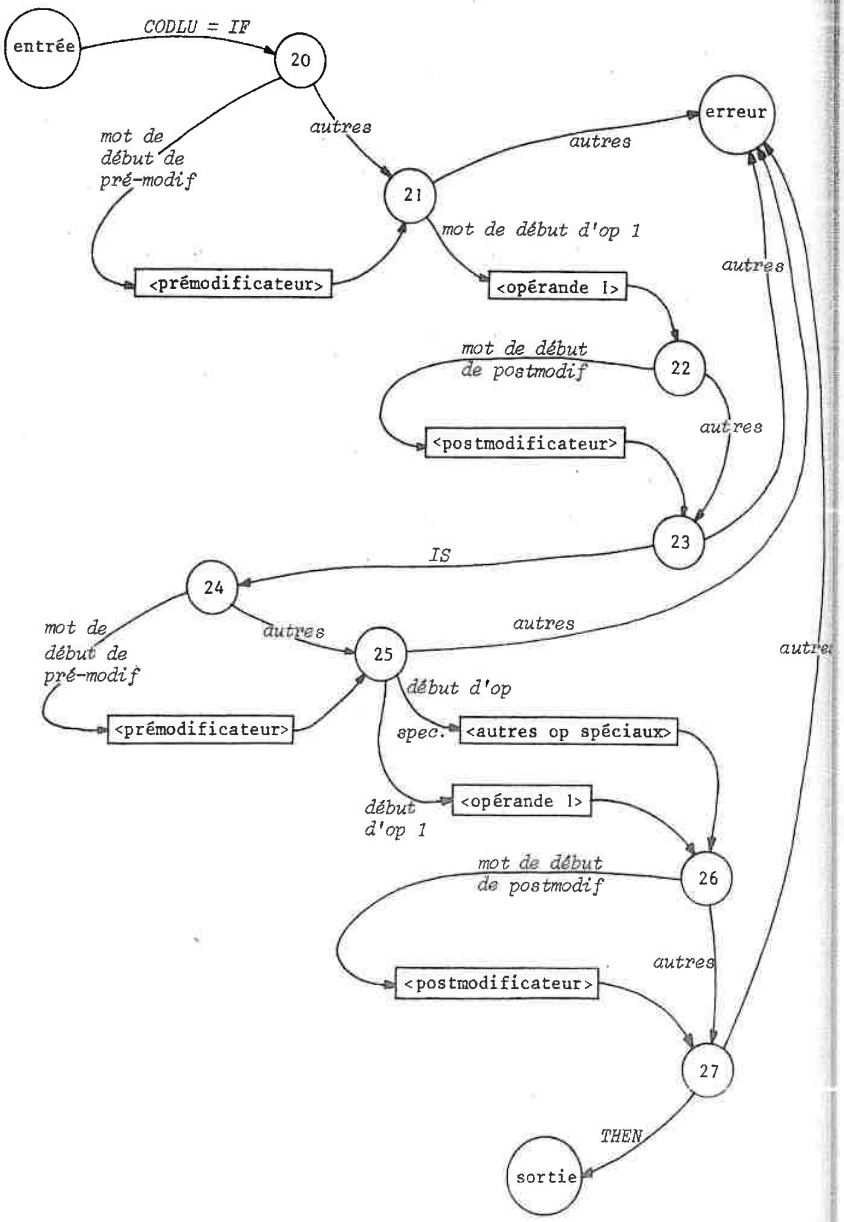


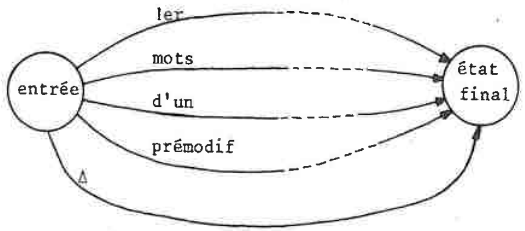
PLANCHE F : exemple d'automate de syntaxe, cas des enquêtes, simplifié (les N° d'état sont arbitraires).

cette branche n'existe pas et le fait de ne pouvoir faire aucun des changements d'état donnés fait passer dans l'état erreur.

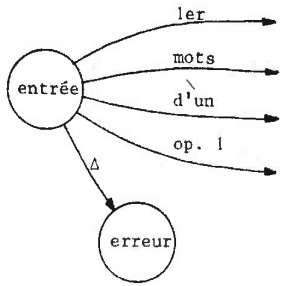
Il en résulte qu'il faudra conserver non seulement des numéros d'état pour sortir de l'état erreur, mais aussi la pile des points de retour des sous-automates.

Exemple :

- le sous-automate <prémodificateur> commencera par :



- le sous-automate <opérande 1> commencera par :



Notons que lorsqu'on rencontre un Δ dans l'automate, on fait le changement d'état qu'il impose sans rechercher le CODLU suivant, puisque le CODLU courant n'a pas été utilisé ; il le sera pour le changement d'état suivant (sauf si on rencontre un autre Δ).

3312 - Représentation de cet automate en mémoire

On le représentera à l'aide d'un ensemble de sept tableaux utilisés en parallèle : CARLU, NEXTR, SSAUTO, DERNR, FINAL, TACHE et NUMMES.

L'ensemble des changements d'états possibles à partir d'un état donné est défini comme suit :

- i positions consécutives de CARLU contiennent les i valeurs de CODLU qui provoquent un changement d'état (autre que vers l'état erreur)
- la position de DERNR correspondant au i ème CODLU permis est mise à VRAI pour indiquer que ce CODLU est le dernier permis. Si le CODLU courant n'a pas été trouvé lorsqu'on rencontre DERNR(i) = VRAI, on passera à l'état erreur
- NEXTR(j) contient soit le prochain numéro à employer pour indicer les tableaux lorsque CODLU = CARLU(j) (état suivant), soit le numéro à employer à la sortie d'un sous-automate s'il y en a un
- SSAUTO(j) = indice de début d'un sous-automate auquel on doit se débrancher si CODLU = CARLU(j) (ou 0 s'il n'y en a pas)
- FINAL(j) = VRAI lorsqu'on est arrivé à un état final d'un sous-automate
- NUMMES(j) est le numéro du message-erreur qui devra être imprimé lors d'un passage dans l'état erreur (il n'y aura donc de numéro dans NUMMES(j) que si DERNR(j) = VRAI).

De plus, TACHE(j) contient le numéro du sous-programme traitant la sémantique et la génération du code lorsqu'on a trouvé dans le texte à analyser le mot dont le code est en CARLU(j).

On appellera "règle de N° NOREGL" l'ensemble des sept éléments CARLU(NOREGL), NEXTR(NOREGL), SSAUTO(NOREGL), DERNR(NOREGL), FINAL(NOREGL) et NUMMES(NOREGL).
NOREGL est initialisé à 1 au début de l'analyse syntaxique : la règle numéro 1 correspond donc à l'état initial de l'automate.

Exemple : la portion de l'automate de la page 76 sera représenté par les règles :

N° d'ordre	CARLU	NEXTR	SSAUTO	DERNR	FINAL	TACHE	NUMMES
1	nom du prog.	2	0	V	F	20	54
2	:	3	0	V	F	21	55
3	↵	8	0	F	F	0	0
4	Δ	5	0	V	F	4	0
5	↵	8	0	V	F	0	56
6	suite						

Au départ, NOREGL = 1. L'analyseur lexical envoie le premier mot du texte ; si ce mot est un nom de programme, on a CARLU(NOREGL) = CODLU et on applique la règle, c'est à dire :

- on exécute la tache N° 20 (TACHE(NOREGL))
- NOREGL devient NEXTR(NOREGL) donc 2, car SSAUTO(NOREGL) = 0
- on demande le CODLU suivant et on continue avec la nouvelle valeur de NOREGL.

Si au départ, le premier CODLU ne correspondait pas à un nom de programme, on essayerait d'augmenter NOREGL de 1 pour trouver CODLU, jusqu'à ce que DERNR(NOREGL) soit VRAI. Or, dans le cas présent, cette condition est réalisée dès le départ ; on passe donc dans l'état erreur, avec pour numéro de message NUMMES(NOREGL) = 54. On détaillera ce processus dans le § 333.

Si SSAUTO(NOREGL) ≠ 0 :

- on exécute la tâche TACHE(NOREGL)
- on empile NEXTR(NOREGL)
- on continue en NOREGL = SSAUTO(NOREGL), jusqu'à ce qu'on soit amené à exécuter une règle où FINAL(NOREGL) = VRAI
- à ce moment, on donne à NOREGL la valeur qui est au sommet de la pile, et on continue.

Si l'on exécute une règle telle que FINAL(NOREGL)=VRAI mais que la pile est vide, on ignore l'indication état final. Il est donc possible de traverser des sous-automates en séquence, sans les appeler.

3313 - Algorithme en pseudo-algol de l'analyseur syntaxique

La pile s'appelle PILSYN et son pointeur NIVO ; ANCREG est initialisé à 1 avant le premier appel et permet une reprise en cas d'erreur.

```
ANALYSE SYNTAXIQUE :
début
AIG = FALSE ;
RECHERCHE :
tant que DERNR(NOREGL) = FALSE faire
      si CODLU = CARLU(NOREGL) alors aller en TROUVE
      sinon NOREGL = NOREGL + 1 ;
commentaire : on n'a pas trouvé CODLU ;
si CARLU(NOREGL) ≠ Δ alors aller en TRT-ERREUR ;
commentaire : il y a un débranchement dans la grammaire, à l'aide du Δ ;
si TACHE(NOREGL) ≠ 0 alors TRTACH(TACHE(NOREGL)) ;
si ERREUR alors aller en RETOUR ;
commentaire : est-ce la sortie d'un sous-automate ? ;
si FINAL(NOREGL) et NIVO ≠ 0 alors
      début
      dépiler NOREGL de PILSYN ;
      aller en RECHERCHE
      fin du cas sortie d'un sous-automate ;
commentaire : y a-t-il une entrée dans un sous-automate ? ;
si SSAUTO(NOREGL) ≠ 0 alors
      début
      empiler NEXTR(NOREGL) sur PILSYN ;
      NOREGL = SSAUTO(NOREGL) ;
      aller en RECHERCHE
      fin du cas débranchement vers un sous-automate
sinon début
      NOREGL = NEXTR(NOREGL) ;
      aller en RECHERCHE
      fin du cas pas de sous-automate ;

TROUVE : commentaire : on a trouvé la bonne règle ;
si TACHE(NOREGL) ≠ 0 alors TRTACH (TACHE(NOREGL)) ;
```



```
si ERREUR alors aller en RETOUR ;
commentaire : est-ce la sortie d'un sous-automate ;
si FINAL(NOREGL) et NIVO ≠ 0 alors
      début
      dépiler NOREGL de PILSYN ;
      aller en TERMINE
      fin du cas sortie d'un sous-automate ;
commentaire : y a-t-il une entrée dans un sous-automate ? ;
si SSAUTO(NOREGL) ≠ 0 alors
      début
      empiler NEXTR(NOREGL) sur PILSYN ;
      NOREGL = SSAUTO(NOREGL)
      fin du cas entrée dans un sous-automate
sinon NOREGL = NEXTR(NOREGL) ;
TERMINE : ANCREG = NOREGL ; commentaire : sauvegarde pour le cas d'erreur ;
aller en RETOUR ;


TRT-ERREUR : commentaire : on essaye de corriger automatiquement (cf § 333) ;
TRERR(code 3) ;
si NOT ERREUR alors
      début
      NOREGL = ANCREG ;
      aller en RECHERCHE
      fin ;
RETOUR : fin de l'analyseur syntaxique, on passe au  CODLU suivant.
```

- Notons que, compte tenu du nombre de règles qu'il a fallu définir pour représenter STRATOS (environ 500 en ménageant 1/4 de place libre pour d'éventuelles mises à jour ultérieures), il suffit de..... 9 bits

pour représenter un NEXTR et de..... 9 bits
pour un SSAUTO

- FINAL et DERNR étant booléens, il leur suffit de 2 * 1 bit.... 2 bits

- il y a un peu plus de 100 tâches, donc NOTACH occupe..... 7 bits
au minimum

- de même pour les messages erreurs (NUMMES)..... 7 bits
- on verra que les CODLU occupent..... 9 bits

ce qui fait au total : 43 bits au minimum.

Un mot sur UNIVAC 1110 faisant 36 bits, il faudra au moins 2 mots par règle ; de cette façon, la grammaire occupe 1 000 mots, soit environ 1K mots.
On a choisi le découpage suivant :

CARLU sur 36 bits (car on fait une recherche séquentielle dans CARLU).
NEXTR et SSAUTO sur 9 bits chacun
DERNR et FINAL sur 1 bit chacun
TACHE et NUMMES sur 8 bits chacun
} = 36 bits

Remarques :

- deux tâches particulières assurent la compilation des directives et des instructions (COMDIR et COMINS)
- on trouvera un listing complet de la grammaire en annexe A.

332 - Organisation des tables de noms

STRATOS n'est pas un langage nécessitant de nombreux noms de variables fournis par le programmeur ; en effet, ses principaux opérandes sont des entités définies sur la molécule en cours de traitement (ou sur le motif d'un automate de stratégie). Or, comme dans ALCHEM, on se réfère par rapport à la numérotation des atomes (que l'on peut afficher sur l'écran ou qui correspond à la numérotation standard du motif) : on parlera de ATOM 4 ou BOND 1-2 etc...
Quant aux variables arithmétiques qu'on peut être amené à employer, on les définira comme dans ALCHEM, soit par un nom qui leur est propre (PRIORITY, PROXIMITY, ELECTRONEGATIVITY...) ou par VALUE n, qui désigne le registre n.

Il en résulte que les seuls noms que devra définir l'utilisateur seront les étiquettes des instructions GO TO.

Par contre, toutes les phrases devant s'exprimer à l'aide de phrases chimiques (anglaises)aussi naturelles que possible, il y aura un très grand nombre de noms réservés (la liste des atomes du tableau périodique

des éléments et des groupes fonctionnels en comptent 103 à eux seuls).

On va donc choisir une méthode de gestion de tables qui soit à la fois très rapide en recherche et en mise à jour, pour qu'on puisse traiter avec aussi bien la table des étiquettes (temporaire et très variable par définition) que celle(s) des noms réservés (permanente et peu sujette à variation).

Une première solution a été adoptée pour accélérer cette gestion de tables : on va employer plusieurs tables physiquement distinctes :

- une qui regroupe tous les noms employés au niveau de la grammaire : approximativement, ceux des enquêtes et les premiers mots des directives et des instructions. On l'appellera TAB1ER.
- une qui contient les autres noms des directives et des instructions, qui ne sera employée que par les sous-programmes de compilation de ces phrases : TABDIV
- une pour les étiquettes : TABLAB
- une pour les noms de groupes d'atomes employés dans certaines directives relatives au phosphore (voir le § 427) : TABGRA.

Ces quatre tables ont la même structure ; nous l'étudierons donc en général. TAB1ER, TABDIV et TABGRA sont stockées sur disque,chargées en début de compilation et ré-écrites à la fin seulement si elles ont été modifiées (voir § 333 sur les erreurs). Au contraire, TABLAB est entièrement définie en cours de compilation, et détruite à la fin.

Ces tables ont été organisées à l'aide d'un hash-code avec chaînage. Soit TABNOM la table en cours d'utilisation : elle est découpée en trois sous-tables :

- TNOM (double-mots), qui contient les noms
- POINT (mots), qui contient les chaînages
- TCOD (mots), qui contient les codes (CODLU) des noms.

La fonction de hashing est calculée comme suit :
soit MOTLU le nom à traiter ; il est sur un double-mot (MOTLU1 - MOTLU2).
On calcule :

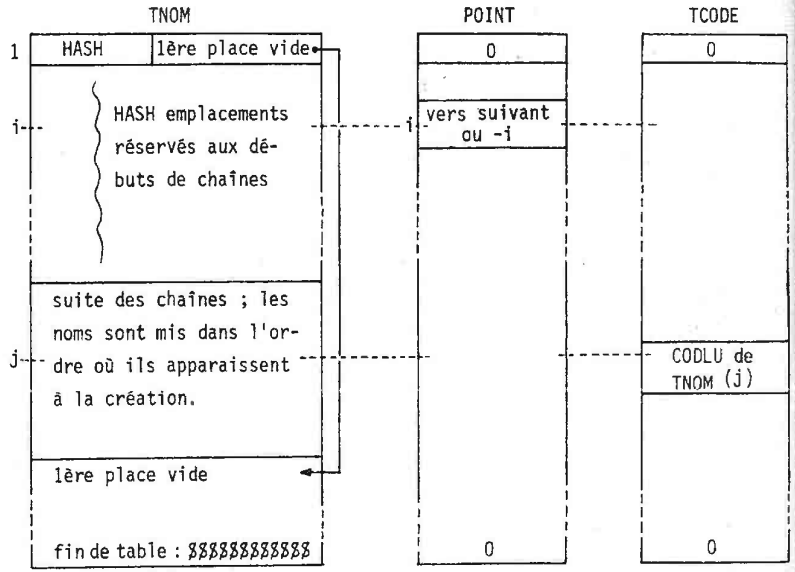
H = reste de la division de |XOR(MOTLU1 , MOTLU2) | par HASH,

où HASH est un nombre premier proche du tiers de la taille de la table (système employé entre autres par des compilateurs PL/1 : voir (19)).
La valeur de H est l'indice dans TABNOM du premier mot rencontré avec ce

reste ; la table POINT permet de définir des chaînages en cas de collision. On peut espérer ainsi trouver un nom (ou être sûr qu'il est absent) après environ 3 ou 4 comparaisons. L'emploi du chaînage permet de contrôler aisément sur un listing la répartition et la taille des chaînes. Par exemple, dans le cas de TAB1ER (258 noms) :

37, 5 % des noms sont accessibles avec une seule comparaison
26 % avec 2 comparaisons
20 % " 3 "
11 % " 4 "
4 % " 5 "
4 noms en nécessitent 6 et un seul en nécessite 7.

On a voulu construire une fonction dont le résultat est l'indice d'un MOTLU donné dans la table demandée en paramètre ou l'indice de l'emplacement où il doit être chaîné s'il est absent. C'est autour de cette fonction unique HCODE (TNOM, POINT) que sont bâtis les sous-programmes de création, de mise à jour ou d'utilisation des tables. L'organisation exacte qu'il a fallu adopter pour cela est la suivante :



Au retour de la fonction HCODE, on obtient :

HCODE > 0 : le nom cherché existe, à l'indice HCODE
HCODE = 1 : le mot cherché n'est pas un nom (nombre ou caractère spécial)
HCODE < 0 : le nom cherché n'existe pas, il peut être rajouté à la chaîne qui se termine en -HCODE.

Lors de l'emploi de HCODE, un aiguillage (ABS1 pour "ABsent ⟹ 1") permet de faire HCODE = 1 si le nom n'existe pas dans la table. Le code d'un tel nom sera donc toujours 0 (TCOD(1)), qui n'est jamais employé , et il y aura erreur.

Une fonction RAJMOT permet de rajouter un (ou plusieurs) noms dans les tables. On lui donne en paramètre :

- les tables concernées
- le nom à rajouter
- son code.

Par un appel à la fonction HCODE, elle peut déterminer si ce nom est déjà présent dans la table demandée (HCODE > 1). Sinon, la valeur absolue de HCODE donne l'endroit où se terminait le chaînage auquel le nouveau nom doit être rattaché ; il est ainsi possible de prolonger ce chaînage, le nouveau nom étant rajouté en fin de table.

Une fonction RAJSYN permet de rajouter un synonyme à un nom déjà présent (le père). On lui donne en paramètre :

- le père
- le synonyme (le fils).

RAJSYN essaye successivement les deux tables TAB1ER et TABDIV. Un premier appel à HCODE pour le père permet de savoir si le père est présent ; s'il est présent, on prend son code, puis on appelle RAJMOT pour le fils avec ce code.Au cas où le père est absent des deux tables, ou bien si le fils est déjà présent dans une table, RAJSYN imprime un message-erreur.
On a donc RAJMOT = TRUE lorsque l'adjonction a pu se faire normalement.

Notes :

- le code est en fait composé de deux parties. En effet, plusieurs mots peuvent avoir le même code vu par la grammaire (par exemple tous les VERBE-DIR), mais il faudra quand même pouvoir les différencier dans les tâches ; le sous-code est ACTION.

- la table des labels étant modifiée en cours d'analyse d'une ligne, elle devra faire partie des variables à sauver en début de ligne pour une reprise éventuelle en cas d'erreur (voir § 333).

333 - Traitement des erreurs

Le souci était double à ce niveau : d'une part proposer un système de correction d'erreurs aussi évolué que possible (grâce aux possibilités offertes par le conversationnel en particulier), mais surtout, éviter autant que possible les risques d'erreurs.

Or, les phrases du langage sont des phrases chimiques aussi proches que possible du langage courant. Il ne s'agit toutefois pas à proprement parler d'un problème de communication en langue naturelle, puisqu'on n'utilise qu'un sous-ensemble relativement réduit et assez simple à définir du langage courant : les phrases anglaises qui permettent d'énoncer une action chimique déterminée s'expriment rarement de plusieurs façons très différentes.

Il en résulte que l'apprentissage d'un tel langage sera relativement aisé pour un chimiste, tout au moins en ce qui concerne la syntaxe des directives, instructions et questions dans les enquêtes.

Les principes retenus sont les suivants :

1) souplesse du vocabulaire

Si un nom possède un synonyme ou une abréviation couramment employé, ou encore une autre orthographe possible par trop voisine (par exemple, le mot français équivalent), alors ce synonyme (etc...) est également placé dans les tables, avec le même CODLU ; toute l'analyse syntaxique étant faite sur les CODLU, il y aura transparence de ces synonymes. De plus, l'utilisateur peut lui-même rajouter des synonymes à sa convenance ; dans ce cas, les tables sont ré-écrites sur disque en fin de compilation pour que les nouveaux noms soient conservés.

Exemples :

ATOM, ATOME, ATOMS et ATOMES sont synonymes
REDUCE et DECREASE sont synonymes
STEREOCHEMISTRY et STEREO sont synonymes.

Une amélioration possible à ce principe consisterait à faire rajouter automatiquement par le système tout nom fréquemment tapé à la place d'un autre ;

mais il ne semble pas, d'après des observations réelles, que cette possibilité supplémentaire, qui peut être assez coûteuse, soit vraiment indispensable.

2) souplesse des phrases

Tout mot dont la présence n'est pas indispensable pour la compilation d'une phrase peut être omis sans dommage.

Exemple : la phrase par laquelle on demande de rajouter une charge positive à l'atome 5 s'écrit :

ADD + AT ATOM 5	elle peut aussi être écrite
ADD + AT 5	ou même
ADD + 5	ou
ADD + ATOM 5.	

En effet, dans cette phrase, seuls les mots ADD, + et 5 ont une réelle signification à conserver, puisqu'on ne peut pas rajouter une charge électrique à autre chose qu'à un atome.

3) correction automatique

Il résulte de tout ceci que, les phrases comportant de nombreuses redondances, et les seuls mots à employer obligatoirement ayant peu de risque d'être mal employés par un chimiste, les erreurs les plus courantes que l'on va rencontrer seront des erreurs de frappe (ce fait est d'ailleurs confirmé par des observations faites durant les essais du compilateur). On va donc inclure au système de traitement d'erreurs un sous-programme de correction automatique (voir 24,25 et 26) qui sera à même de corriger des fautes du type :

- il manque une lettre dans un nom, corrigé si le début du nom et la lettre qui suit sont semblables : ALCPHOL et ALCOHOL
- il y a une lettre en trop, corrigé si le début du nom et les deux lettres suivantes sont semblables : ALCOOHOL et ALCOHOL
- deux lettres sont interverties, corrigé si le début du nom et la lettre qui suit sont semblables : ACLOHOL et ALCOHOL
- deux mots sont accolés, corrigé s'il reste au moins un blanc au bout de MOTLU (sinon le 2ème nom pourrait avoir été tronqué au milieu) et si on arrive à former un 1er nom valide avec le début de MOTLU.

Notons qu'on sait au retour de la fonction HCODE, si le mot appartient ou non à la table ; on déclarera donc d'emblée non corrigeable

un mot existant que la grammaire n'autorise pas à cet endroit du texte.

On pourra introduire dans le sous-programme de correction automatique une table supplémentaire (cf. DAMERAU (28)), constituée comme suit : on associe à chaque nom des tables un compteur qui donne le nombre de lettres, et un mot-code de 36 bits (un mot-mémoire sur Univac) dont chaque bit est associé à un signe possible dans les noms :

bit Ø ⟶ Ø
(
bit 9 ⟶ 9
bit 1Ø ⟶ A
(
bit 35 ⟶ Z

Le bit i de ce mot est mis à 1 si le signe correspondant est employé dans le nom ; par exemple pour ATOM, les bits 1Ø, 22, 24 et 29 seront à 1.

A l'entrée dans le module de correction, on calcule le mot-code du nom à corriger. On n'essayera la correction automatique que sur les noms de la table dont la longueur est égale ou différente de 1 et dont la distance de Hamming (30) est au plus égale à 2.

On gagne ainsi du temps en recherche préliminaire. Mais il faut noter que la correction de deux mots accolés ne sera pas possible avec cette méthode.

De toutes les façons, l'utilisateur reste seul maître, le système lui demandant de juger sa proposition de correction d'orthographe. Il faut noter par ailleurs qu'il y a des limites à un tel système, toute erreur de frappe ne pourra pas être rattrapée (s'il y en a plusieurs...).

4) mode débutant et mode savant

En mode savant, on imprime simplement un message-erreur bref, et le numéro du message explicite. En mode débutant, on imprime en plus le message explicite. Le choix entre ces deux modes est fait par une option en début de compilation ; il est possible de faire un appel au secours lorsqu'on est en mode savant pour obtenir un message plus explicite à un moment donné.

Il reste à pouvoir corriger "manuellement" une erreur qui ne peut pas l'être automatiquement. On a retenu trois manières de le faire :

* 1) retaper complètement une nouvelle ligne ; cette méthode, peu pratique en conversationnel, est celle qui est utilisée implicitement en batch. Son principal inconvénient est qu'on a autant de risque de faire une erreur de frappe en retapant qu'en tapant la première fois

* 2) si l'erreur est due uniquement au mot qui la provoque, on peut modifier simplement ce mot ; l'analyse reprend alors sur ce mot

* 3) dans tous les autres cas, on peut corriger en rajoutant du texte, supprimant un mot ou remplaçant un mot par un nouveau texte.

Rappelons qu'on peut également corriger en introduisant un synonyme.

Pour que ceci soit possible, on conserve l'état du compilateur à chaque début de ligne, c'est à dire la pile d'analyse syntaxique, le numéro de règle à utiliser et toute autre variable susceptible de changer en cours d'analyse et devant servir plus tard (par exemple, dernier N° employé dans une CLD... voir § 334, ou table de labels). Il sera donc possible de reprendre l'analyse d'une nouvelle ligne (ou de l'ancienne corrigée) comme si la ligne erronée n'avait pas existé.

Lorsqu'apparaît une condition d'erreur, on entre dans un sous-programme,

- qui positionne un aiguillage ERREUR à VRAI
- tente une correction automatique s'il y a lieu
- sinon, ou si la correction automatique s'avère impossible ou est refusée, imprime un message erreur
- appelle le sous-programme de lecture qui se charge des corrections demandées par l'utilisateur (en effet, c'est lui qui charge le buffer d'entrée... il a donc semblé logique de lui confier ce rôle).

Lorsqu'on revient du sous-programme d'erreur,

- ou bien on a ERREUR = FAUX (correction automatique faite par exemple) et on peut continuer avec ce mot corrigé là où on s'était arrêté,
- ou bien on a ERREUR = VRAI (l'utilisateur propose une correction, il faut l'analyser), et on sortira de tous les sous-programmes où l'on pourrait être entré, jusqu'à arriver dans le programme principal à la section qui ré-initialise l'état du compilateur en début de ligne. On peut alors reprendre l'analyse.

Le sous-programme est divisé en six parties distinctes, qui traitent chacune un cas précis :

- erreur code 1 = erreur de syntaxe détectée par l'analyseur syntaxique (grammaire) ou par l'analyseur lexical (caractère erroné). Le code 2 est réservé pour ces erreurs au niveau de la structure IF-THEN-ELSE-BEGIN-DONE des enquêtes.

- erreur code 3 = erreur à un endroit où plusieurs noms sont permis mais où celui qui est fourni est impossible (essai de correction automatique).

- erreur code 4 = même chose s'il n'y a qu'un nom possible (on envisage de le trouver par correction automatique).

- erreur code 5 = erreur impossible à essayer de corriger automatiquement.

- erreur code 6 = erreur sur un nombre dont la valeur donnée est impossible (p. ex. N° de groupe = 1 ou 2) : on redonne simplement la nouvelle valeur. Il est aussi possible de demander l'emploi de corrections standards (retaper la ligne...).

Exemple (ce qui est tapé par l'utilisateur est précédé de >) :

```
                                    commentaires :
>1) BREAK BOND 3-4                  ---- directive correcte
>2 ADD P+ AT ATOM 5                 ---- il manque ) après le 2
     ^                              ---- place de l'erreur
 CARACTERE OU MOT ERRONE
 ***CORRECTION ?
 ERR 61                             ---- N° du message explicite
 LA SYNTAXE DEMANDE ')'             ---- message explicite
>*MODIF                             ---- demande de correction
 ***MODIF ?
 2 ADD P+ AT  ATOM 5                ---- ré-écriture de la ligne pour corriger
>+)!                                ---- correction : insérer ) après le 2
 ***MODIF ?
 2) ADD P+ AT ATOM 5                ---- ligne obtenue
>      -                            ---- nouvelle correction : supprimer le
 ***MODIF ?
 2) ADD + AT ATOM 5                 ---- ligne obtenue
>                                   ---- fin des corrections, l'analyse repar
>3) BREAK BOND 3,4,5,6              ---- directive suivante
                ^
```



```
 CARACTERE OU MOT ERRONE
 ***CORRECTION ?
 ERR 24
 LE CYCLE A PLUS DE SOMMETS QUE VOUS EN AVEZ DECLARE
 OU: PLUS DE 10 SOMMETS
 OU: PLUS D'UN NUMERO (OU COUPLE) POUR DESIGNER UNE LIAISON
 RETAPEZ OU *MODIF
>*MODIF                             ---- demande de correction
 ***MODIF ?
 3) BREAK BOND 3,4,5,6
>       =RING!                      ---- correction:remplacer BOND par RING
 ***MODIF ?
 3) BREAK RING 3,4,5,6              ---- ligne obtenue
>                                   ---- fin des corrections
>
```

334 - Les combinaisons logiques de directives (ou CLD)

La CLD sera la partie principale d'un programme de stratégie interactive (si elle ne le compose pas entièrement). C'est à l'aide d'une CLD que l'utilisateur va définir ses objectifs, donc les modifications qu'il voudrait apporter à la molécule en cours d'étude.

On n'étudiera pas dans ce chapitre la syntaxe des CLD, qui sera vue au chapitre IV, mais la manière dont est créée et représentée la "table de vérité" (déjà mentionnée au chapitre II).

3341 - Problème général

Rappelons brièvement les résultats obtenus dans la chapitre II. L'utilisateur donne une liste de directives reliées entre elles par les opérateurs logiques courants (AND, OR, XOR et la négation), avec possibilité de parenthésage. On a opté pour la solution consistant à considérer cette expression comme une expression logique dont les variables booléennes représentent les directives. Au cours de la compilation, cette expression logique

va être mise sous forme disjonctive minimale, et les directives seront remplacées par leur code généré ; chaque "variable logique" de l'expression devient en fait un pointeur vers le code généré de la directive correspondante. La CLD se trouvera donc représentée par une suite de "monômes" dont tous les termes (directives) sont reliés entre eux par ET, et qui sont reliés l'un à l'autre par OU. On aura défini ainsi une suite de sous-objectifs qui permettent, chacun, d'atteindre l'objectif décrit par la CLD.

Pour cela, on va développer l'expression logique à l'aide d'un algorithme classique d'exécution d'une expression parenthésée (sans priorité entre les opérateurs, cf. chapitre IV), l'exécution d'un opérateur étant en fait une réunion ou une intersection de monômes. Il est à noter que si toutes les directives employées sont différentes, le développement donnera tout de suite la forme disjonctive minimale, puisqu'aucune mise en facteur en vue de simplifications (par les théorèmes de Morgan (31)) n'est possible. Par contre, dès qu'on emploie plusieurs fois la même directive (ou un XOR, puisque A XOR B = (A AND NOT B) OR (NOT A AND B), ce qui introduit une duplication de A et de B), on risque d'obtenir une expression non minimale. Dans ce cas, on en fera la simplification par la méthode de minimisation programmable de Quine-MacCluskey (32) ou de Harris (33).

3342 - Traitement du parenthésage

Etudions maintenant le processus du développement. Le premier problème rencontré est celui du parenthésage de l'expression. En effet, nous verrons dans la chapitre IV (§ 423 sur les CLD), qu'on a voulu éviter à l'utilisateur de devoir prévoir le parenthésage, et qu'il lui est possible de le définir au fur et à mesure. Pour cela, on a adopté les conventions suivantes :

- toutes les directives sont numérotées
- on ferme une parenthèse en indiquant le numéro de la directive où doit se placer la parenthèse ouvrante correspondante.

Exemple (deux directives qui se suivent sont implicitement reliées par l'opérateur AND) :

```
1) ADD + AT ATOM 5
2) BREAK BOND 3-4     )1
```

signifie

(ADD + AT ATOM 5 AND BREAK BOND 3-4).

Il en résulte qu'on crée une parenthèse ouvrante par directive, et que ces parenthèses ne seront pas nécessairement fermées. Il y aura donc une opération préalable d'épuration de la CLD.

Un second problème a été posé par le fait que les "variables logiques" de l'expression sont en réalité les directives. Or le code généré (pas plus que la source) d'une directive occupe une place variable en mémoire (en pratique : de 1 à 5 mots). Il n'est donc ni rentable, ni pratique d'employer le code généré des directives pour les représenter dans le développement de la CLD (surtout si la même directive longue revenait plusieurs fois). De plus, il doit y avoir compilation des directives au fur et à mesure qu'elles apparaissent dans la structure logique de l'expression.

Les deux raisons exposées ci-dessus ont conduit au développement de la méthode suivante :

* 1) on génère en début de CLD une directive particulière, que nous appellerons "directive-CLD", et qui annonce une CLD. Son usage exact sera vu plus loin.

* 2) à mesure que l'utilisateur fournit la CLD, on crée d'une part la liste du code généré des directives employées (en prenant garde aux doubles éventuels), complétée d'une table (TIDIR) qui donne l'adresse des directives en fonction de leur rang d'apparition (TIDIR(i) = adresse du code de la i ème directive rencontrée).

* 3) d'autre part, on crée une représentation de la CLD dans laquelle on ne conserve que les parenthèses ouvrantes réellement employées, ainsi que le nombre de fois où elles servent, et dans laquelle les directives sont remplacées par leur rang dans la CLD (donc par leur indice dans TIDIR).

On peut alors envisager d'appliquer sur cette représentation un algorithme classique de développement d'expression parenthésée, ce qui constituera une deuxième phase.

Etudions un exemple de cette première phase :

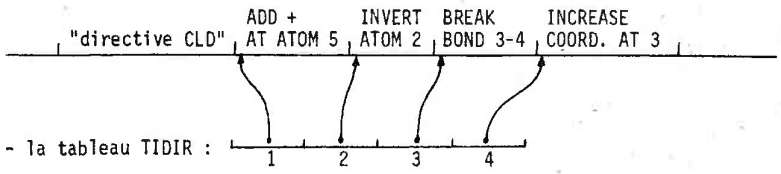
Soit la CLD :

```
1) ADD + AT ATOM 5
2) INVERT ATOM 2
3) BREAK BOND 3-4      )1
      OR
4) INCREASE COORDINENCE AT ATOM 3
5) ADD + AT ATOM 5     )4
```

Cette CLD représente l'expression suivante :
(1 AND 2 AND 3) OR (4 AND 1), où les numéros représentent le rang d'apparitio des directives correspondantes.

Au fur et à mesure que l'utilisateur donne sa CLD, le compilateur génère :

- dans le tableau du code généré :



- la tableau TIDIR : 1 2 3 4

- le nombre de directives, NBDIR = 4
- le tableau TABDIR qui représente la directive :

```
 (   N° 1   utilisée 1 fois
DIR  N° 1   sans négation
AND
 (   N° 2   utilisée 0 fois
DIR  N° 2   sans négation
AND
 (   N° 3   utilisée 0 fois
DIR  N° 3   sans négation
 )
OR
 (   N° 4   utilisée 1 fois
DIR  N° 4   sans négation
AND
 (   N° 5   utilisée 0 fois
DIR  N° 1   sans négation
 )
FINCLD
```

Ceci représente bien l'expression donnée si l'on ne tient pas compte des parenthèses non utilisées. On rappelle que le numéro des directives est l'indice de l'élément de TIDIR qui contient leur adresse dans le code généré.

Algorithme de création de TABDIR :
chaque élément de TABDIR (indicé par ICLD) peut contenir :

- soit une parenthèse ouvrante :

(	N° de la (, qui précède chaque directive	nombre de fois où elle est fermée (initialisé à 0)

- soit un opérateur : AND, OR ou XOR
- soit une directive :

DIR	rang de la directive (indice dans TIDIR)	divers : négation et ONLY (voir plus loin)

- soit une parenthèse fermante :

)		flag ONLY

- soit l'indication de fin de directive .

La création de TABDIR est faite par des tâches appelées par la grammaire,

- lors de la rencontre d'une "parenthèse ouvrante", c'est à dire de la numérotation d'une directive (tâches PODIR, PODIR1 = 1ère parenthèse ouvrante et PODIRE = parenthèse suivant une directive, demandant d'abord un AND implicite à rajouter)
- à la fin d'une directive (tâche FINDIR)
- à la rencontre d'une "parenthèse fermante" (tâche PFDIR)
- à la rencontre de OR ou XOR (tâches OPOR et OPXOR).

On trouvera maintenant une description de ces tâches.

⊕ PODIR1

- préparer la "directive CLD" dans le code généré ; on conserve son adresse dans ADRCLD pour la compléter plus tard
- TIDIR(1) = IGEN : pointe sur la première place vide du tableau de code généré
- NBDIR = 1
- ANCNO = 0 : permet de vérifier la séquence des numéros de directives
- L2DIR = LXOR = FALSE : pour l'emploi de directives identiques ou de XOR (si l'un des deux est TRUE, il faudra minimiser)
- tâche PODIR normale.

⊕ PODIR

- si N° de directive ⩽ ANCNO, erreur (code 6)
- TABDIR(ICLD) = parenthèse ouvrante (non encore utilisée)
- ANCNO = numéro de la directive
- ICLD + 1

⊕ PODIRE

- si N° de directive ⩽ ANCNO, erreur (code 6)
- TABDIR(ICLD) = opérateur AND ; ICLD + 1
- suite comme PODIR

⊕ OPOR et OPXOR

- TABDIR(ICLD) = opérateur OR ou XOR ; ICLD + 1
- dans OPXOR : LXOR = TRUE

⊕ PFDIR

on veut fermer une parenthèse de numéro NB donné. Il faut donc supprimer celle de numéro supérieur qui n'ont pas servi, et rajouter 1 à l'indicateur d'utilisation de celle dont le numéro est égal à NB.

Algorithme en pseudo-algol :

début
PFDIR :
sauver TABDIR dans STDIR pour cas d'erreur de numérotation ;
REPRISE :
i = ICLD1 ; commentaire : voir ci-dessous ;
tant que TABDIR(i) ne représente pas la parenthèse ouvrante de numéro NB faire
début
si TABDIR(i) est une parenthèse ouvrante de numéro inférieur à NB alors
aller en ERREUR ;
i = i - 1 ;
si i = 0 alors aller en ERREUR
fin de la recherche de la parenthèse ouvrante de numéro NB ;
ajouter 1 au 3ème champ de TABDIR(i) ; commentaire : la parenthèse a servi une fois de plus ;
ICLD1 = i-1 ; commentaire : ICLD1 pointe sur la dernière parenthèse réellement ouverte pour éviter les parenthésages "croisés" ;
TABDIR(ICLD) = parenthèse fermante ;

ICLD = ICLD + 1 ;
aller en RETOUR ;
ERREUR :
TRERR (code 6) ; commentaire : on fait corriger le numéro NB ;
rétablir TABDIR d'après STDIR ;
aller en REPRISE ;
RETOUR : fin de PFDIR

⊕ FINDIR

il faut regarder si la nouvelle directive n'a pas déjà été créée dans la CLD, afin de lui affecter le même numéro dans TIDIR, donc de ne la garder qu'en un seul exemplaire. Si une directive re-sert, on aura L2DIR = TRUE. Notons que l'indication de la négation est dans TABDIR et non dans la directive. IGEN indice le tableau du code généré.

Algorithme en pseudo-algol :

début
FINDIR :
si ERREUR dans la compilation de la directive alors aller en RETOUR ;
NBDIR = NBDIR + 1 ;
TIDIR(NBDIR) = IGEN ;
i = 1 ;
tant que directive i ≠ directive NBDIR faire i = i + 1 ;
si i ≠ NBDIR alors
début commentaire : la nouvelle directive existait déjà ;
NBDIR = NBDIR-1 ;
IGEN = TIDIR (NBDIR) ;
L2DIR = TRUE
fin de la suppression de la nouvelle (on pointe avant) ;
TABDIR(ICLD) = directive de numéro i (éventuellement négative) ;
ICLD = ICLD + 1 ;
RETOUR :
fin de FINDIR.

Remarque importante : les variables

NBDIR (nombre de directives rencontrées)
ADRCLD (adresse de la "directive CLD", qui doit être complétée à la fin)

ANCNO (dernier numéro de directive pour contrôler la séquence)
ICLD1 (niveau atteint dans la suppression de parenthèses ouvrantes inutiles)
LXOR ET L2DIR (indicateur de demande de minimisation),
ainsi que le tableau TABDIR et TIDIR avec leurs indices ICLD (et NBDIR), font partie de l'état du compilateur à sauvegarder à chaque début de ligne pour permettre la reprise en cas d'erreur.

3343 - Développement effectif

Jusqu'ici, on a simplement obtenu une nouvelle représentation de la CLD, dans un format interne plus pratique à employer. Il reste à la développer. On définit pour cela :

- PANDE = pile des opérandes
- PEUR = pile des opérateurs
- les opérations EMPILER, DEPILER, et SOMMET.

Dans l'expression à développer, qui se trouve maintenant dans TABDIR, les opérateurs sont AND, OR, XOR, '(' dont le nombre d'utilisations n'est pas nul ')' et l'indicateur de fin de CLD, et les opérandes sont soit une directive (représentée par son numéro annoncé par le code "DIR" dans TABDIR) soit une sous-expression (somme de monômes logiques)déjà traitée.

L'algorithme du développement est le suivant (LCOUR est le code de TABDIR(ICLD)) :

```
pour chaque ICLD variant depuis 1 tant que TABDIR(ICLD) ≠ FINDIR faire
    début
    extraire LCOUR de TABDIR (ICLD) ;
    aiguillage sur LCOUR :
        - opérande : EMPILER (PANDE,LCOUR) ;
        - opérateur ou fin d'expression :
              si initialisation alors EMPILER (PEUR, LCOUR)
              sinon début
                    si SOMMET(PEUR) ≠ '(' alors EXECUTER
                    sinon si LCOUR ≠ fin d'expression alors FINIR
                          sinon EMPILER(PEUR, LCOUR)
                    fin ;
        - '(' : EMPILER (PEUR, LCOUR) ;
        - ')' : tant que SOMMET (PEUR) ≠ '(' et PEUR  non vide faire  EXECUTER
```



```
              si PEUR n'est pas vide alors DEPILER (PEUR,O)(O sera perdu,
                                                          c'est la '(')
              sinon erreur
    fin
```

Définition de EXECUTER :

```
EXECUTER : DEPILER (PANDE,Y) ;
           DEPILER (PANDE,Z) ;
           DEPILER (PEUR, O) ;
           appliquer l'opérateur O sur X et Y, résultat R ;
           EMPILER (PANDE, R).
```

Initialisation de cet algorithme : on empile dans les piles respectives jusqu'au premier opérateur AND, OR ou XOR compris.
Notons que cet algorithme n'a pas besoin en fait de vérifier les cas d'erreurs, puisqu'ils ont déjà été testés pendant la construction de TABDIR.

3344 - Représentation des monômes

A partir d'ici, comme on l'a déjà vu un peu plus haut, les "variables logiques" employées comme opérandes dans cet algorithme ne sont plus seulement des directives, mais des sommes de monômes, sous-expressions qui se construisent peu à peu. En effet, l'exécution d'un opérateur logique ne donne pas ici un résultat "vrai" ou "faux" ; il s'agit en fait de l'opération correspondante sur des ensembles (réunion, intersection et différence symétrique pour AND, OR et XOR).

Il reste à trouver un moyen de représenter ces ensembles (en fait, ces sommes de monômes logiques).

On sait que si on a NBDIR variables, il y aura au plus 2^{NBDIR} monômes possibles, chaque monôme pouvant être le produit logique d'au plus les NBDIR variables avec ou sans négation. On appellera directive (ou variable) négative une directive (ou variable) employée avec négation (DO NOT) ; par opposition, on parlera de directive (ou variable) positive s'il n'y a pas de négation.

On voudrait donc obtenir l'expression sous sa forme disjonctive minimale ; or on sait que dans de très nombreux cas (lorsqu'il n'y a pas de directives identiques ni de XOR), l'expression sera tout de suite minimale (d'après expérience, ce sera la majorité des cas). Il en résulte qu'on a in-

têrêt à choisir une représentation telle qu'il soit possible de distinguer les trois possibilités suivantes :

- une directive est positive
- " " " négative
- " " " n'est pas employée dans le monôme; elle y sera donc indifférente.

La solution qui consisterait à développer entièrement l'expression sous forme disjonctive normale, dont le grand avantage consiste à représenter les monômes sous forme de chaînes de bits sur lesquelles les opérations logiques (ou plutôt ensemblistes) sont simples et rapides, présente en fait l'inconvénient d'exiger systématiquement une minimalisation. Le gain de temps engendré par l'emploi des opérations logiques normales sur les monômes est donc très largement compensé par la perte due au développement et à la minimalisation. De plus, le résultat de la minimalisation étant une expression sous forme disjonctive "non-normale", il faudrait de toutes façons un changement de représentation en cours de route , pour pouvoir représenter les trois états cités ci-dessus.

On se tournera donc vers une représentation permettant de les distinguer dès le départ. Ainsi, une variable logique seule sera en fait le monôme où cette variable sera notée positive ou négative, et toutes les autres indifférentes.

De plus, il faudra pouvoir appliquer les règles de l'algèbre de Bool et les théorèmes de Morgan. En particulier :

a - $A.\overline{A} = 0$

b - $A.A = A$

c - $A.0 = 0$, d'où : monôme quelconque . 0 = monôme 0

d - $A + 0 = 0$, d'où : somme de monômes quelconques + monôme 0 = la somme inchangée

e - de plus, une variable indifférente étant en fait la somme logique d'elle-même en positif avec elle-même en négatif (son complément), on aura : variable A . elle-même-indifférente = A.

On constate donc qu'en plus des états positif, négatif et indifférent, il faudra pouvoir représenter l'état "nul" ; toutefois cet état sera temporaire, puisque la présence d'une seule variable nulle rend le monôme nul, et que les monômes nuls sont transparents dans une somme de monômes.

On a donc quatre états à représenter par variable ; c'est pourquoi on essayera de représenter chaque variable logique sur 2 bits. Un monôme serait ainsi une succession de groupes de 2 bits. Or on constate par expérience qu'une CLD comptera la plupart du temps un petit nombre de directives : 4 à 6 environ en moyenne, rarement plus de 10. En effet, il s'agit de représenter une opération chimique élémentaire très particulière et surtout très limitée. Il est donc tout à fait possible de représenter un mônome sur un mot-mémoire, les opérations logiques (et autres) seront ainsi réduites au minimum. Un mot-mémoire comportant 36 bits sur Univac 1110, on serait limité à 18 directives dans une CLD, ce qui met à l'abri de tout risque. Il en résulte qu'une somme logique de monôme serait en fait une suite de mots-mémoire (réunion ensembliste).

C'est donc en étudiant le fonctionnement du AND entre deux monômes qu'on va essayer de trouver une représentation pratique pour une variable logique sur 2 bits.

Un monôme est l'ensemble des NBDIR groupes de 2 bits représentant chaque variable ; dans l'ensemble représentant l'intersection (AND) de deux monômes , les NBDIRS groupes de 2 bits doivent être construits à partir des groupes correspondants de chaque monôme, de telle sorte que les règles a,b,c et e soient respectées. La solution suivante permet d'y arriver :

- variable indifférente : 00
- " positive : 01
- " négative : 10
- " nulle : 11.

De cette façon, l'intersection de deux monômes sera obtenue, paradoxalement, en faisant l'opération OR :

- règle a : $(A.\overline{A} = 0)$ 01 OR 10 = 11
- règle b : (A.A = A) 01 OR 01 = 01
  ou 10 OR 10 = 10
- règle c : (A.indifférent = A) 01 OR 00 = 01
  ou 10 OR 00 = 10.

Un monôme sera nul dès qu'apparaît la configuration 11 dans un bloc de 2 bits au moins (règle c).

Exemple :

(monôme $A\overline{B}$) AND (monôme AC) représenté par
(01 10 00) OR (01 00 01) = (01 10 01) : on trouve bien la représentation du

monôme $A\bar{B}C$.

Pour déterminer la présence de 11 dans un monôme, deux solutions sont possibles :

* 1) tester 2 bits par 2 bits

* 2) faire un et logique entre l'ensemble des bits de rang pair et celui des bits de rang impair du monôme ; si le résultat n'est pas nul, le monôme comportait 11 et il est nul. Pour cela :

- M_1 = monôme AND masque composé de groupes de 2 bits tous à 01
- décaler les bits du monôme d'une position vers la droite (on verra que ceci est possible sans perte de bit, car le groupe de droite sera toujours 00 ou 10 : voir § 3345 : "ONLY")
- faire M_2 = monôme-décalé AND même-masque
- enfin M_1 AND M_2, qui doit être nul.

Des essais comparatifs entre ces deux méthodes n'ont pas permis de déceler de différence de temps (au millième de seconde).

3345 - Représentation de ONLY

On a vu, au chapitre II, qu'on désire pouvoir demander qu'une directive (ou plusieurs reliées par AND) soit seule à être exécutée, donc qu' aucune autre action que celle qu'elle demande ne soit faite sur la molécule. Il faut donc pouvoir représenter ONLY au niveau de chaque monôme. Or dans un monôme marqué ONLY, avant minimalisation, il ne doit pas y avoir de variables indifférentes, par définition ; elles devront toutes être marquées négatives (10 au lieu de 00).

Mais ceci est insuffisant. En effet, lorsque la CLD comporte NBDIR variables, seules ces NBDIR variables sont représentées dans les monômes; considère donc toutes les autres variables possibles comme indifférentes. De fait en l'absence de ONLY, elles le sont réellement car on accepte d'autres modifications sur la molécule et en particulier celles qui y correspondent.

Il résulte de tout ceci qu'on doit rajouter à chaque monôme une indication sur l'emploi de "toutes les autres directives", qui correspond à l'emploi de ONLY.

On restera cohérent avec la notation décrite précédemment en adoptant la règle suivante : on rajoute un NBDIR + une ième groupe de 2 bits, positionné à 00 en l'absence de ONLY (toutes les autres sont indifférentes)

et à 10 lorsqu'il y a ONLY (toutes les autres sont négatives)

Remarques :

- lors de la minimalisation éventuelle, des simplifications entre monômes marqués ONLY peuvent introduire des variables indifférentes
- les deux bits les plus à droite étant ainsi 00 ou 10, la méthode de repérage des 11 (monômes nuls) par décalage est bien possible.

3346 - Réalisation pratique du développement par STRATOS

A partir de cela, il ne reste plus qu'à appliquer l'algorithme de développement déjà décrit et les théorèmes de Morgan sur la représentation donnée ci-dessus pour obtenir l'expression sous forme d'une somme de monômes logiques :

- à la rencontre d'une directive dans TABDIR (LCOUR = "DIR"), on crée un monôme qui ne contient que la directive correspondante, et LCOUR devient son indice dans le tableau SIGMON (SIGma de MONômes).
- à la rencontre d'un AND, on applique le théorème de Morgan :

$(M_1 + M_2 + \ldots)$ AND $(M'_1 + M'_2 + \ldots) = (M_1$ AND $M'_1) + (M_2$ AND $M'_2) + \ldots$

l'opération AND étant un OR logique entre deux mots suivi du repérage des 11.

- à la rencontre d'un OR, on concatène les suites de mots-mémoire représentant les sous-expressions déjà obtenues en supprimant les doubles :

$(M_1 + M_2 + \ldots) + (M'_1 + M'_2 + \ldots) = M_1 + M_2 + \ldots + M'_1 + M'_2 + \ldots$

- à la rencontre d'un XOR, on applique la définition

A XOR B = A AND $\bar{B}$ + $\bar{A}$ AND B.

Le XOR sera l'opération la plus longue à cause des complémentations : $\overline{(M_1 + M_2 + \ldots)} = \bar{M}_1.\bar{M}_2. \ldots$

$\overline{(v_1.v_2. \ldots)} = \bar{v}_1 + \bar{v}_2 + \overline{\ldots}$

Tout ceci sera fait dans la tâche FINCLD appelée lorsqu'on rencontre une phrase ne pouvant plus faire partie d'une CLD (voir remarque en fin de §).

Θ FINCLD

2ème phase = développement de la CLD à partir de TABDIR :

- TABDIR(ICLD) = '.'
- développement
- si L2DIR ou LXØR =.TRUE. , simplification

- mise en place de la "table de vérité" de la CLD : NBDIR, ISIGM, TIDIR (1,..., NBDIR), SIGMØN(1,..., ISIGM)
- fin de la génération de la "directive CLD", à l'adresse ADRCLD

Ø Sous-programme EXECØP

il sert à exécuter un AND, un ØR ou un XØR. L'opérateur est au sommet de PEUR et les opérandes sont les deux derniers de PANDE ; le résultat ira dans PANDE. Les opérandes (et le résultat) sont en fait les indices des sommes de monômes dans SIGMØN ; on utilise le tableau RSIGM (indicé par IRSIGM) pour mettre une somme de monômes-résultat intermédiaire.

Appelons : Y = PANDE(IPANDE) = pointeur vers l'opérande droite
Z = PANDE(IPANDE-1) = pointeur vers l'opérande gauche
O = PEUR(IPEUR) = opérateur

Ø Exécution d'un ET

début
IRSIGM = 1 ;
pour i variant de Z à Y-1 faire
pour j variant de Y à ISIGM-1 faire
début
RSIGM(IRSIGM) = SIGMON(i) OR SIGMON(j) ;
s'il n'y a pas de couples de bits à 11 alors IRSIGM = IRSIGM + 1
fin ;
recopier RSIGM dans SIGMON en éliminant les doubles éventuels ;
ISIGM = Z + IRSIGM-1 ;
fin de l'exécution d'un ET.

Voir exemple sur la planche G.

Ø Exécution d'un OU

début
k = Y ;
pour j variant de Y à ISIGM-1 faire
début
pour i variant de Z à Y-1 faire
si SIMON(j) = SIGMON(k) alors aller en DOUBLE ;
SIGMON(k) = SIGMON(j) ;

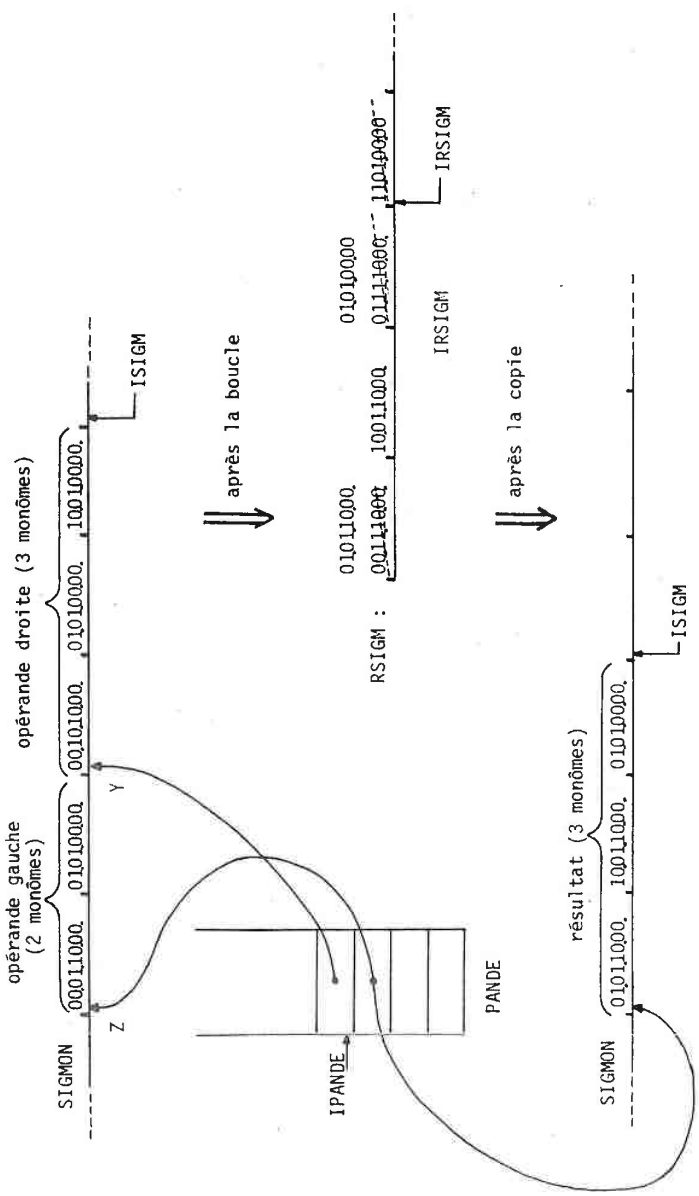


PLANCHE G : schéma de l'exécution d'un AND.

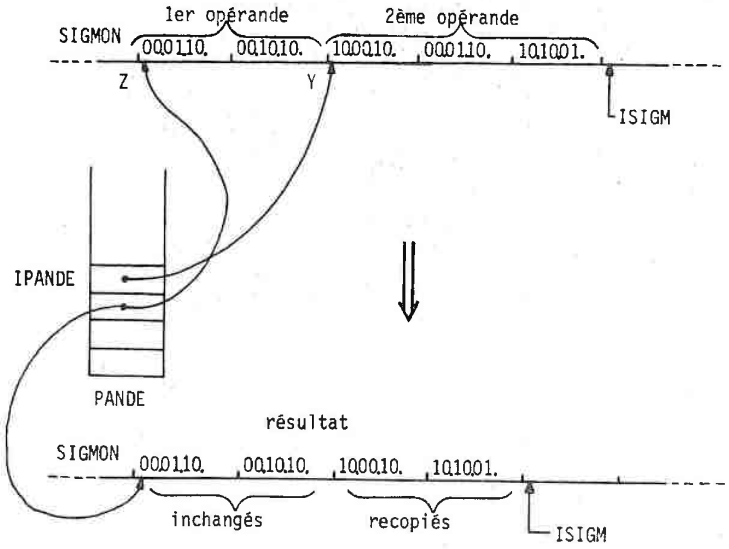
K = K+1

DOUBLE :

fin de la boucle sur l'opérande droite ;

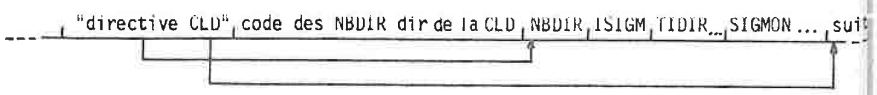
fin de l'exécution du OU.

Exemple :



⊕ Exécution de XOR et simplification : non encore programmés

3347 - En conclusion, nous voyons qu'une CLD sera représentée dans le tableau du code généré, de la manière suivante :



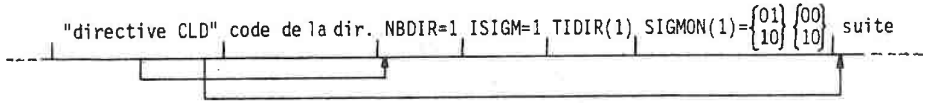
La "directive CLD" sert donc à indiquer l'adresse qui suit la dernière directive, et l'adresse de la première instruction ne faisant plus

partie de la CLD. Elle ne pourra être complétée que tout à la fin de compilation de la CLD, d'où ADRCLD (voir plus haut).
ISIGM est la dimension de SIGMON.
SIGMON est la forme disjonctive minimale de l'expression : c'est la "table de vérité" dont il était question au chapitre II.

Remarques :

- après programmation de la minimalisation, il faudra faire des essais chronométrés afin de voir si cette opération est réellement rentable. En effet, les méthodes de Quine-MacClushey ou de Harris étant très coûteuses en place mémoire, il est pratiquement sûr qu'il faudra travailler sur disque (eu égard à la place déjà occupée par le reste du programme).

- lorsqu'une CLD est réduite à une seule directive, la syntaxe demande de ne pas la numéroter, ce qui serait parfaitement inutile (cf. chapitre IV). Cette directive unique aura malgré cela la structure d'une CLD, afin de simplifier l'interprétation de code généré, qui reste ainsi standard :



- en fait, FINCLD est exécutée non pas dès qu'on a trouvé un mot ne pouvant plus faire partie de la CLD, mais si en plus ce mot est un premier mot de phrase possible. En effet, si l'on fait une erreur sur le premier mot d'une ligne de CLD, il ne faut pas la considérer comme terminée, car la nouvelle ligne, erronée, devrait en faire partie. Deux solutions sont possibles : laisser s'exécuter FINCLD en incluant dans l'ensemble des variables à sauver en début de ligne, toutes celles qui sont modifiées par cette tâche ; ou introduire un indicateur booléen qui montre que FINCLD est susceptible d'être exécutée, l'exécution effective étant retardée après que l'analyseur syntaxique ait accepté le premier mot de la nouvelle ligne. C'est cette dernière solution qui a été retenue, parce que l'exécution de FINCLD risque d'être relativement longue, surtout s'il y a minimalisation.

335 - Compilation des directives et des instructions

On a vu que les directives et les instructions sont compilées "mot à mot" par deux tâches particulières : COMDIR et COMINS respectivement.

Pour l'analyseur syntaxique (grammaire), une directive est constituée du mot "VERBE-DIR" et une instruction du mot "VERBE-INSTR". Il en résulte que CODLU n'est pas suffisant pour distinguer les directives (ou les instructions) entre elles. La table des codes des premiers mots contient donc en fait deux codes : CODLU et ACTION, qui sert dans ces cas (voir § 332).

On entre dans COMDIR et COMINS en pointant sur le premier mot de la phrase (le verbe), on en sort en pointant sur le premier mot qui ne peut plus en faire partie.

Ces deux tâches sont écrites sous forme de deux sous-programmes séparés, qui commencent par un aiguillage sur ACTION renvoyant à une séquence bien séparée pour chaque directive ou instruction.

Le code généré pour les directives et les instructions est une succession de champs de longueur et de signification variables. Deux d'entre eux sont toujours présents :

- le champ 1, d'un seul bit, distingue les directives (1) des autres types de phrases (0)
- le champ 2, de 6 bits, contient le code opération.

Au fur et à mesure de l'étude des phrases, on crée les divers champs du code dans une succession de mots-mémoire, en binaire : c'est le tableau INFO.

Lorsque la compilation de l'instruction ou de la directive est terminée, le sous-programme SCYFR compacte le tableau INFO en fonction de la longueur des champs, donnée dans un tableau qui sera utilisé en parallèle avec INFO : si le i ème champ est à coder sur j bits, le i ème élément du tableau des longueurs devra contenir j, et SCYFR extraira les j bits de droite de INFO(i), en vérifiant que les autres sont nuls.

Exemple : soit la directive ADD - AT ATOM 3. Son code généré est formé des champs suivants :

- champ 1	1 bit	bit directive	→	0
- champ 2	6 bits	code opération	→	000001
- champ 3	1 bit	inutilisé	→	0
- champ 4	1 bit	+(0) ou -(1)	→	1
- champ 5	8 bits	numéro d'atome	→	00000011

Le tableau des longueurs devra donc contenir les valeurs 1,6,1,1,8.
INFO sera chargé au fur et à mesure avec les valeurs 0,1,0,1,3.
SCYFR va donc créer le mot code généré suivant :

0	000001	0	1	00000011	0 jusqu'au bout

Note : une directive ou instruction objet occupe un nombre entier de mots-mémoire, SCYFR assure le passage d'un mot au suivant si un champ ne loge plus.

Le passage d'un mot de la phrase à compiler au suivant est assuré par le sous-programme SMOTLU qui fournit (en particulier) :

- MOTLU (12 caractères) qui contient le nouveau mot (nombre, caractère spécial ou nom)
- NB = valeur en binaire si MOTLU contient un nombre
- ALPHA = VRAI si MOTLU contient un nom
- NUM = VRAI si MOTLU contient un nombre.

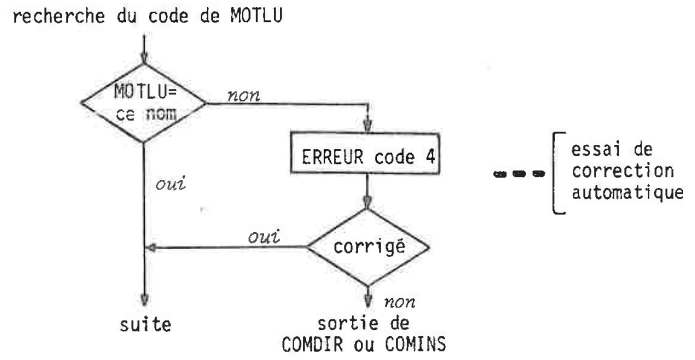
ALPHA et NUM sont donc tous deux FAUX si MOTLU contient un caractère spécial.

On rappelle que tous les noms permis dans les directives et les instructions, à l'exclusion de leur premiers noms, sont rassemblés dans une table à part : TABDIV = table des noms divers.

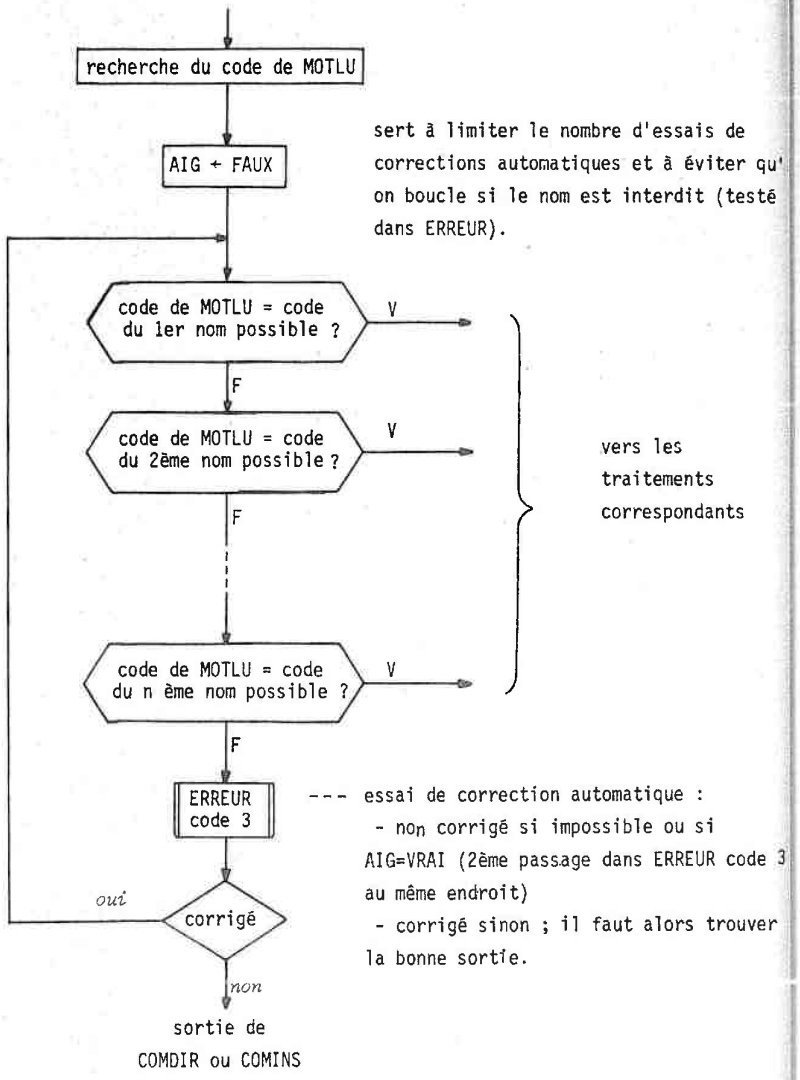
Chaque fois qu'on a obtenu un nouveau mot de la phrase à analyser, on en cherche le code dans TABDIV grâce à un appel à HCODE, puis on compare ce code à ce qui est permis à ce niveau de la phrase. Les indications ALPHA et NUM permettent de séparer facilement les cas. On peut constater de toutes manières que le nombre de choix possibles est rarement élevé (cf. § 427 et 428 qui donnent la liste des directives et des instructions).

Après l'emploi de ALPHA et NUM, deux cas principaux peuvent se présenter lorsque ALPHA = VRAI :

* 1) un seul nom est permis à cet endroit ; on trouvera alors la structure suivante :

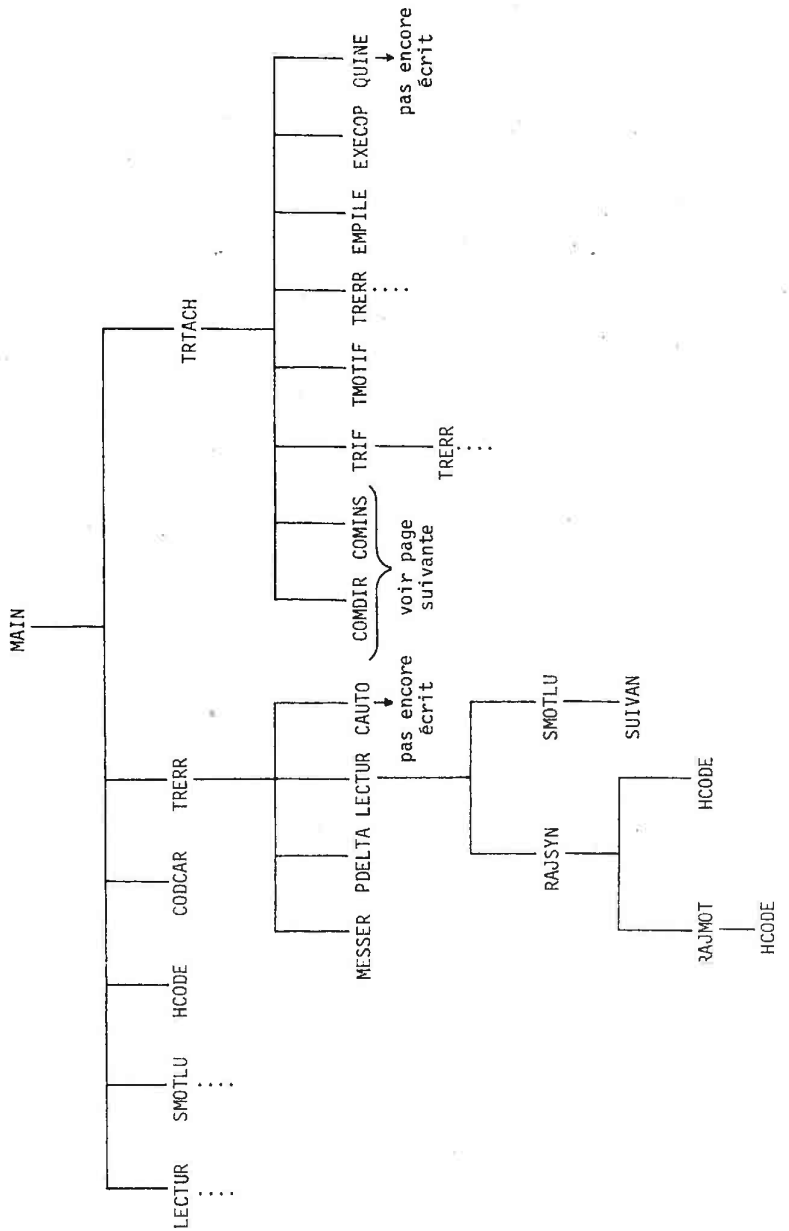


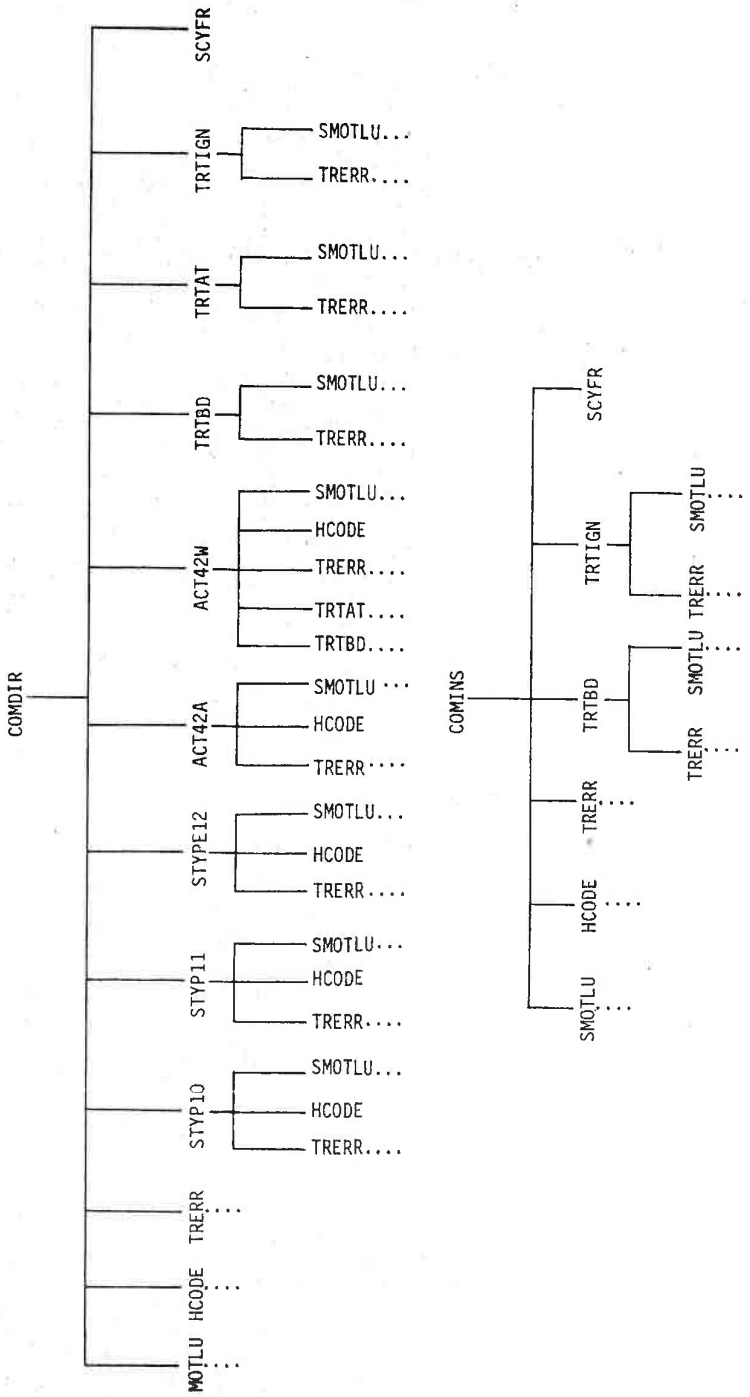
* 2) on peut rencontrer plusieurs noms à cet endroit ; on aura alors la structure suivante



Note : on rappelle que les synonymes ne sont pas traités de cette façon : ils ont le même code dans la table.

336 - Structure générale de STRATOS - rôle des sous-programmes



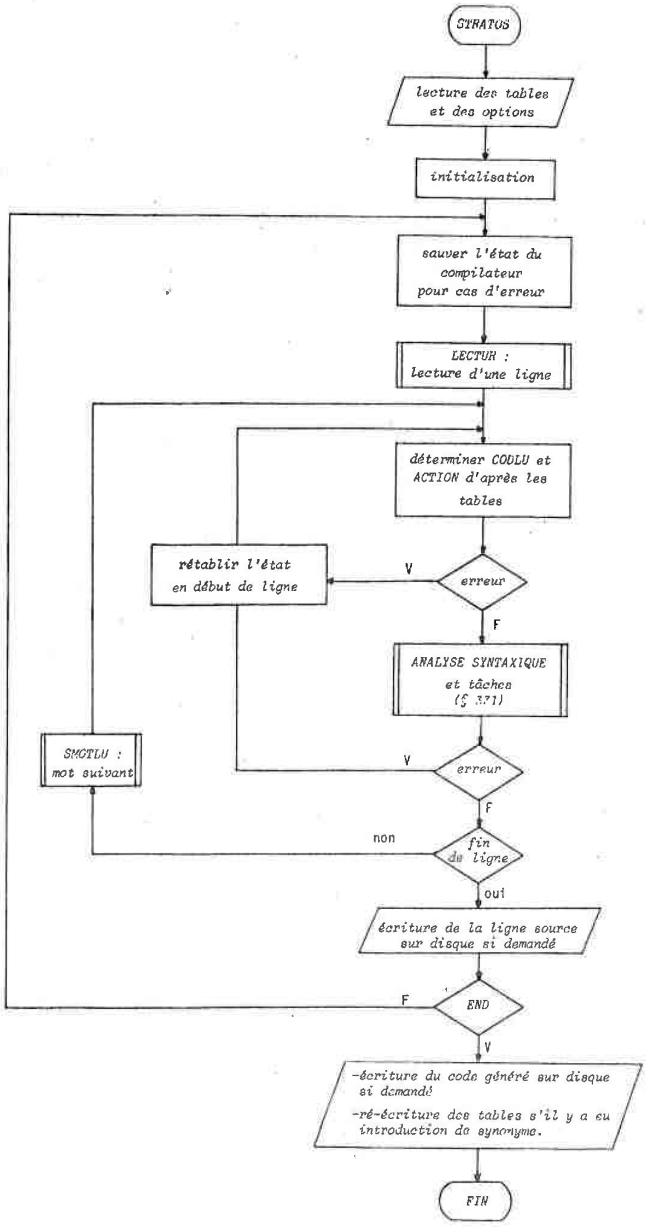


Rôle des divers sous-programmes :

MAIN	programme principal, analyse syntaxique (grammaire)
CODCAR	fournit le code des caractères spéciaux
LECTUR	lit une nouvelle ligne, traite les corrections et positionne sur le premier mot de la ligne
SMOTLU	fournit le mot suivant de la ligne, zéro en fin de ligne
SUIVAN	interne à SMOTLU ; passe au caractère suivant
HCODE	recherche un nom dans une table
RAJMOT	rajoute un nouveau nom dans une table
RAJSYN	" " " synonyme dans les tables
TRERR	traitement d'erreurs
MESSEX	écriture du message-erreur
PDELTA	place le signe ^ sous le mot faux
CAUTO	essai de correction automatique
TRTACH	traitement des tâches d'après la grammaire
COMDIR	compile les directives
STYP10	(sous type 1-0) partie'AT ATOM n'des directives de type 1
STYP11	(sous type 1-1) option'TO n'de certaines directives de type 1
STYP12	(sous type 1-2) partie'TO SBP'.. de la directive MODIFY STRUCTURE
ACT42A	partie'AT ATOM n'de la directive MAKE groupe (ACTion 42-At)
ACT42W	partie'WITH.'.. de la directive MAKE groupe (ACTion 42-With)
TRTBD	compile une liste de numéros de liaisons (BOND ou RING)
TRTAT	" " " " " d'atomes
TRTIGN	" la directive (ou instruction) IGNORE
COMINS	" les instructions
TRIF	tâches particulières des enquêtes
TMOTIF	compile un motif chimique d'automate de stratégie
SCYFR	compacte les champs du code objet obtenus dans INFO
EMPILE	empile un opérateur ou opérande logique dans le traitement des CLD
EXECOP	exécute un opérateur logique dans le traitement des CLD
QUINE	simplifie la CLD.

Un fichier sur disque (STRATAB) contient les tables :

- la grammaire (GRAMMR)
- la table des noms employés dans la grammaire (TAB1ER)
- " " " " " " COMDIR et COMINS (TABDIV)
- " " " groupements d'atomes de la directive EXCHANGES(TABGRA)



c'est le sous-programme de traitement d'erreur qui demande les lignes de correction.

PLANCHE H : organigramme général de STRATOS.

- la table des messages-erreurs explicites.

Les quatres premières sont chargées en mémoire en début de compilation et elles y résident ; les deux lères tables de noms (TAB1ER et TABDIV) sont réécrites à la fin s'il y a eu adjonction de synonymes.

Au contraire, la table des messages-erreurs reste sur disque. En effet, la place étant comptée en mémoire, on a préféré cette méthode à devoir découper le compilateur en phases. La table est organisée par adresse, chaque message étant placé sur un secteur (28 mots de 6 caractères), sous forme de format FORTRAN. Le numéro du message permet donc de l'adresser à une constante additive près.

Il reste à donner un organigramme général de STROTOS (voir planche H).

En conlusion de ce chapitre, nous dirons que le langage ainsi défini devrait permettre non seulement aux utilisateurs non informaticiens de s'exprimer face au système avec un minimum de contraintes, mais aussi aux personnes chargées d'en assurer les extensions futures, de réaliser leur tâche avec aussi peu de difficultés que possible. De plus, la définition du langage se voulant dès le départ la plus large et la plus souple possible, on peut s'attendre à un faible travail de maintenance lors des premières extensions projetées.

CHAPITRE IV

STRATOS DU POINT DE VUE DE L'UTILISATEUR

Ce chapitre est destiné à l'utilisateur non-informaticien. Il suppose une certaine connaissance du langage ALCHEM : voir (13) et (12). Il décrit complètement le langage STRATOS et l'utilisation de son compilateur en mode débutant et en mode savant.

41 - DEFINITIONS

- un programme de stratégie (ou simplement une stratégie) est l'ensemble de toutes les phrases qui sont données au système de stratégie pour qu'il dirige la recherche dans le fichier des transformations afin de proposer des chemins de synthèse.

- le système de stratégie est l'ensemble des programmes informatiques utilisés : perception de la molécule dessinée, compilateur STRATOS et son interpréteur STRATEX, lecteur du fichier...

- la stratégie est donnée sous forme d'une suite de phrases. On y distingue des "enquêtes", des "instructions" et des "directives".

- une enquête est composée d'une question suivie des blocs de phrases décrivant les actions à exécuter suivant la réponse à la question.

- une instruction est une phrase qui décrit une action dont le

résultat n'affecte pas la molécule elle-même : calculs, GO TO, ensembles...

- une directive est une phrase chimique dont le résultat est une action sur la molécule : casser une liaison, modifier la valence d'un atome...

- une combinaison logique de directives (ou CLD) est une suite de directives reliées par des opérateurs logiques AND (toujours implicite), OR et XOR (OR exclusif), négation. Le parenthésage est possible.

Notons qu'une stratégie interactive se réduira pratiquement à une CLD, alors qu'un automate de stratégie utilisera toutes les autres phrases, et peu les CLD.

42 - SYNTAXE

421 - Un mot peut être soit un nombre, soit un caractère spécial (*, ; etc), soit un nom (ATOM, RING etc).

422 - Dans tous les noms qu'on peut rencontrer dans les phrases, seuls les 12 premiers caractères sont pris en compte ; les suivants, s'il y en a, sont ignorés. Le premier caractère d'un nom doit être une lettre, les suivants peuvent être des lettres ou des chiffres uniquement. Dans certains noms particuliers désignant des associations d'atomes avec un atome de phosphore, le signe = est permis (double liaison).

423 - Un nombre ne peut pas avoir plus de 10 chiffres, sinon il est tronqué à droite, et sa valeur numérique n'est plus significative. En pratique, les nombres sont relativement petits (15 registres, 63 atomes...).

424 - Une directive ou une instruction doit être en principe donnée sur une seule ligne (ou carte). Il est possible d'utiliser des lignes-suites en l'annonçant sur la ligne incomplète par le signe & ; tout ce qui suit ce signe jusqu'à la dernière position de la ligne (colonne 72) est alors ignoré et on continue sur la ligne suivante. Il n'est pas permis de couper un mot. Si l'on désire employer plusieurs lignes-suite pour un seul ordre, il y aura un & en fin de chaque ligne, sauf sur la dernière. Le cas des enquêtes sera vu à part.

425 - Il n'y a pas de séparateur spécial, les blancs sont ignorés sauf dans les noms et les nombres. La fin d'un nom est marquée par le premier caractère non alphanumérique rencontré , et la fin d'un chiffre par le premier caractère non numérique.

426 - Toutes les phrases sont données sous forme de phrases chimiques anglaises correctes. Il sera toutefois possible de simplifier ces phrases, certains mots n'étant présents que pour rendre la phrase correcte ; de tels mots pourront être omis.

427 - Liste des directives

Dans cette liste, on adoptera les conventions suivantes :

- un mot souligné est obligatoire, les autres sont optionnels pour rendre les phrases correctes
- des accolades indiquent un choix
- des crochets indiquent une partie de phrase en option
- des points de suspension signalent que le mot (ou le signe, ou l'option...) précédent peut se répéter plusieurs fois.

{ADD / REMOVE} {+ / -} AT ATØM n

INTRØDUCE STEREØCHEMISTRY AT ATØM n

INTRØDUCE groupe AT ATØM n

LØSE STEREØCHEMISTRY AT ATØM n

MAKE RADICAL AT ATØM n

REMØVE groupe AT ATØM n

{REDUCE / INCREASE / MØDIFY} {CØØRDINENCE / VALENCE} FØR ATØM m [TØ n]

MØDIFY STRUCTURE FØR ATØM n TØ { TBP / SBP / SP3 / ØCTAHEDRAL }

MAKE SUBSITUTIØN gra 1 [/ gra 2] AT ATØM n

EXCHANGE groupe 1 [INTØ groupe 2] AT ATØM n

USE TYPE n_1 /n_2 AT ATOM n

Dans ce premier type de directives, les mots AT et FØR sont synonymes ; "groupe", "groupe 1" et "groupe 2" représentent des noms de groupes fonctionnels ; "gra 1" et "gra 2" sont des groupements d'atomes (phosphore).

Liste des noms des groupes fonctionnels utilisables

ACID	PHOSPHATE
ACIDX ou ACID HALIDE	PHOSPHITE
ALCOHOL	PHOSPHINE
ALDEHYDE	PHOSPHINOXIDE
AMIDE	PHOSPHONATE
AMINE	PHOSPHINATE
CYANO ou NITRILE	PHOSPHONITE
OLEFIN ou OLEFINIC ou ALKENE	PHOSPHINITE
EPOXIDE	PHOSPHORYL
ESTER	PHOSPHONIUM
ETHER	PHOSPHONATEX
HALIDE	PHOSPHINATEX
IMINE	PHOSPHONITEX
KETONE	PHOSPHINITEX
NITRO	METHYL
ACETYLENE ou ACETYLENIC ou ALKYNE	PHENYL
CARBONIUM ION	BENZYL
VINYLW	DGROUP
ESTERX	OXO ou OXOGROUP ou CARBONYL
AMIDEZ	XGROUP

CYCLOPROPANE ou CYCLOPROPYL	ZGROUP
IMINEZ	WGROUP

Liste des groupements d'atomes pour MAKE SUBSTITUTION

PO PC PH PN PCL P=O P=S POH PF PX
POR PSH PS

Liste des numéros dans la directive USE

32 33 34 44 54 55 66

{ ARØMATIZE / BREAK / DEARØMATIZE } { BØND { FRØM ATOM n_1 TO ATOM n_2 / N } / RING N_1 , N_2 ... }

{ INCREASE / MAKE / REDUCE / INVERT } BØND N

IZØMERIZE BØND N

MIGRATE BØND N TØ BØND M

Dans tous ces cas, N, N_1, ... sont de la forme n [- n']

On peut donc définir un cycle soit par les numéros des atomes, soit par les numéros des deux atomes limitant les liaisons du cycle, séparés par un tiret.

La taille des cycles est limitée à 10 atomes.

MAKE { RING / r M RING / ≥ 6 M RING / 6 M ± RING } WITH { ATØMS n_1 [, n_2] ... / BØNDS N_1 [, N_2] ... }

Dans l'option r M RING, le nombre d'atomes ou de liaisons doit être ⩽ r ; dans ce cas, on a r ⩽ 6.

Dans les options >6 M RING ou 6 M+ RING (équivalents), le cycl doit comporter entre 7 et 10 atomes ou liaisons.

Dans l'option RING, il doit y avoir au plus 10 numéros d'atomes ou de liaisons.

MAKE groupe { WITH { ATØMS n_1 , n_2 ... / BØNDS N_1 , N_2 ... } [AT ATØM n] / AT ATØM n [WITH { ATØMS n_1 , n_2 ... / BØNDS N_1 , N_2 ... }] }

" groupe" est un groupe fonctionnel.

INVERT ATOMS n_1 [, n_2]...

REARRANGE ATOMS n_1 [, n_2]...

CALL nom-de-réaction [AT ATOM n]

Note : dans toutes ces directives, les noms :

- ATOM, ATOME, ATOMS et ATOMES sont synonymes; de même respectivement
- BOND et BONDS
- STEREO et STEREOCHEMISTRY
- AROMATISE et AROMATIZE
- DEAROMATISE et DEAROMATIZE
- REDUCE et DECREASE
- OCTAHEDRAL et OCTAEDRAL.

428 - Instructions

Un certain nombre d'instructions sont reprises comme elles sont dans ALCHEM. En voici la liste :

⊕ instructions arithmétiques

SUM UP { VALUE / PRIORITY / m } [AND { VALUE n_2 / PRIORITY }] [FOR EACH] (addition)

SUBT { VALUE n_1 / PRIORITY / m } [FROM { VALUE n_2 / PRIORITY }] [FOR EACH] (soustraction)

MULT [{ VALUE n_1 / PRIORITY }] BY { VALUE n_2 / PRIORITY / m } [FOR EACH] (multiplication)

DIV [{ VALUE n_1 / PRIORITY }] BY { VALUE n_2 / PRIORITY / m } [FOR EACH] (division)

Dans toutes ces opérations, si l'un des opérandes est omis, il est pris égal à PRIORITY implicitement (cf. ALCHEM).
L'option FOR EACH est un facteur de répétition de l'instruction : elle sera exécutée une fois pour chaque élément d'un ensemble défini auparavant (voir ci-dessous les ensembles) ; exemple :

IF (2) IS WGROUP THEN SUBT 50 FOR EACH

On soustrait 50 à PRIORITY autant de fois qu'il y a de groupes attracteurs dans l'ensemble (2).

Instructions d'affectation :

SET { STEP / VALUE n_1 / PRIORITY } TO { VALUE n_2 / PRIORITY / m }

affecte la valeur de VALUE n_2 , PRIORITY, ou m (numérique) à STEP, VALUE n_1

ou PRIORITY. STEP est un registre qui contient le nombre d'étapes maximal permis dans l'exécution d'une CLD ; il vaut 1 si on ne lui donne pas d'autre valeur.

SET { VALUE n_1 / PRIORITY } TO COUNT (M)

où COUNT (M) représente le nombre de bits à 1 dans l'ensemble (M).

On rappelle que, comme dans ALCHEM, on dispose de 8 registres pour y stocker des valeurs numériques : VALUE 0 à VALUE 7. VALUE 0 est réservé pour PRIORITY.

⊕ ensembles

DEFINE FRAGMENT (N_1) AS { ATOMS / BONDS } CONNECTED TO { (N_2) / ATOM n / BOND n }

Isole dans l'ensemble (N_1) une portion de molécule.

PUT NOT (L) { AND / OR / XOR } NOT (M) INTO (N)

définit une opération logique entre les ensembles (L) et (M), et place le résultat dans l'ensemble (N).

⊕ conditions de réaction

CONDITIONS [{ SLIGHTLY / STRONGLY }] <C> [OR [{ SLIGHTLY / STRONGLY }] <C>]

où <C> représente l'un des noms suivants :
BASIC, ACIDIC, OXIDIZING, REDUCING, COLD, HOT, ORGANOMETALLIC, PHOTOCHEMICAL, PROTIC, APROTIC, NON POLAR, HYDROGENATING, HALOGENATING, NUCLEOPHILIC, ELECTROPHILIC, AQUEOUS, FLUORINATING, SULFURATING.

⊕ instructions de contrôle

{ GOTO / GO TO } étiquette

où "étiquette" est un nom qui ne doit pas être réservé, et qui indique l'adresse de saut. On définit une étiquette (donc l'adresse) en l'écrivant comme premier mot d'une ligne, suivi du caractère ':'.

STOP permet de définir un arrêt de programme de stratégie, autre que la dernière instruction physique.
Un GO TO ou un STOP doivent être suivis d'une définition d'étiquette ou de END (fin de programme).

KILL permet de tuer le programme de stratégie (par exemple, à la suite d'une enquête, qui constate qu'on est dans un cas d'impossibilité).

END est la dernière ligne d'un programme de stratégie.

429 - La phrase IGNORE

Elle est à mi-chemin entre directive et instruction. Elle sert à définir l'ensemble des liaisons que les transformations ne doivent en aucun cas modifier en cours de stratégie. Or, on sera amené à définir en début de stratégie un tel ensemble, qui doit pouvoir rester valable tout au long de la recherche du chemin de synthèse, d'une étape à l'autre, mais auquel on peut être amené à apporter une modification temporaire en cours de CLD. Il en résulte que cette phrase pourra être employée dans une CLD ou hors d'une CLD, selon les règles suivantes :

- employée hors CLD, IGNORE redéfinit tout l'ensemble des liaisons à ignorer. On l'employera ainsi principalement au début de la stratégie et lorsqu'un changement important de stratégie demande un changement complet de cet ensemble.
- employée dans une CLD, elle met à jour temporairement l'ensemble, jusqu'à ce que soit exécuté le sous-objectif où elle se trouve. L'ensemble est réinitialisé ensuite.

Syntaxe :

IGNORE { $N_1 \left[, N_2\right]...$ / AROMATIC / NOTHING }

les N_i sont de la forme $n_i \left[- n'_i\right]$.

Dans le premier cas, on spécifie une liste de numéros d'atomes, le format n-n' signifiant "les numéros compris entre n et n' ". Les transformations devront laisser inchangées les liaisons qui relient deux atomes indiqués dans la liste.

Exemple :

IGNORE 1 - 5, 7, 8

on ne veut pas toucher aux liaisons reliant deux des atomes de l'ensemble {1,2,3,4,5,7 et 8}.

Si, après cette instruction, on écrit dans une CLD (voir syntaxe en 22B) :

1) IGNORE 12
2) BREAK BOND 10-11)1
OR
3) DO NOT IGNORE 7,8
4) ADD + AT 7)3

alors on ignorera en plus les liaisons qui joignent l'atome 12 aux autres atome de l'ensemble, dans le premier sous-objectif ; par contre on n'ignorera pas celles qui aboutissent sur les atomes 7 et 8 dans le deuxième. Dans tous les cas, l'ensemble de référence est inchangé pour la suite de la CLD.

42A - Les enquêtes

Une enquête s'écrit sous l'une des formes suivantes :

IF question [(N)] THEN
phrase composée

ou

IF question [(N)] THEN
phrase composée
ELSE
phrase composée

ou

IF question (N) ⮌
(non numérique)

Dans ces trois cas, "question" peut prendre l'une des formes suivantes :

- <ensemble 1> { IS / ARE } [NOT] [ALL] <ensemble 2>

- { CATION / ANION / RADICAL / STERIC [-HINDRANCE] / ELENERGY / NLENERGY } { AT / ON } <ensemble 1> { BETTER / WORSE } THAN <ensemble 2>

- <ensemble 1> AND <ensemble 2> ARE [NOT] { CIS / TRANS / EQUIVALENT / PROXIMATE / IN SAME RING }

- <ensemble 1> { IS / ARE } [NOT] ATTACHED TO { Ø / 1 / 2 / 3 / ANY } HYDROGEN

- <ensemble 1> { IS / ARE } [NOT] ON { MOST / LEAST } HINDERED SIDE OF <ensemble 2>

- { ATOM n / (N) } IS [NOT] ATOM IN POSITION n / ATOM P

- question d'enquête numérique, voir plus loin.

Note : la notation (N) représente un numéro d'ensemble (set register).

Définition de <ensemble 1> et <ensemble 2> :

<ensemble 1> s'écrit [<prémodificateur>] <opérande> [<postmodificateur>]

<ensemble 2> s'écrit [<prémodificateur>] { <opérande> / <op 2> } [<postmodificateur>]

ou <postmodificateur>

<prémodificateur> représente l'une des options suivantes :

PRIMARY
SECONDARY
TERTIARY
QUATERNARY
PENTA - COORDINATED

[{ ATOMS / BONDS / GROUPS }] WITHIN { ALPHA / BETA / GAMMA / DELTA / EPSILON } [- { ALPHA / BETA / GAMMA / DELTA / EPSILON }] TO

<postmodificateur> peut s'écrire dans l'un des formats suivants :

ONPATH
OFFPATH
{ ONRING / OFFRING } OF SIZE [OTHER THAN] n [- m]

<op 2> sont des opérandes seulement permis dans <ensemble 2> ; il s'agit de

AROMATIC
ANYWHERE
AXIAL
EQUATORIAL
[{ STRONGLY / SLIGHTLY }] { ELECTROPOSITIVE / ELECTRONEGATIVE }

Quant à <opérande> , on peut écrire :

- opérandes relatifs à des atomes :

ATOM

ATOM { n [IN GROUP { 1 / 2 }] / OTHER THAN n - m }

{ PRIMARY / SECONDARY / TERTIARY / QUATERNARY } ATOM

{ BRIDGEHEAD / ALLYLIC / AROMATIC / NEUTRAL / PATH / RING / SATURATED / { UNSATURATED / SP3 } } ATOM

{ SINGLE / DOUBLE / TRIPLE } BOND ATOM

GROUP { 1 / 2 } ATOM

{ ANION / CATION / CARBON / HETEROATOM / HYDROGEN / HALOGEN / NITROGEN / PHOSPHORUS / OXYGEN / SULFUR / RADICAL / STEREOCENTER / + tous types d'atomes jusqu'à IODINE }

- opérandes relatifs à des liaisons :

BOND

BOND { n-m [IN GROUP {1 / 2}] / OTHER THAN n-m }

{ AROMATIC / CONJUGATED / PATH / RING / UNSATURATED / SATURATED / CISTRANS / NOAROMATIC / AROMATIC } BOND

GROUP {1 / 2} BOND

{ SINGLE / DOUBLE / TRIPLE } BOND

HETEROBONDS

HYDROGBONDS

- les set registers : (1) à (7)
- les groupes fonctionnels.

Exemples de questions :

ATOMS ARE ALPHA TO ATOM 1 IN GROUP 1
ATOM WITHIN ALPHA-BETA TO ATOM 5 IS HYDROGEN
ALCOHOL IS ANYWHERE
(2) IS WGROUP
BOND 3-4 AND BOND 2-5 ARE TRANS

Cas des enquêtes numériques : la question est alors relative à des valeurs numériques, et s'écrit sous la forme :

{ VALUE n_1 / PRIORITY / PROXIMITY / ELECTRONEGATIVITY / VALENCE AT { ATOM n_3 [IN GROUP n_4] / (N) } } IS [NOT]

{ GREATER THAN / ≥ / LESS THAN / ≤ / EQUAL TO / = } [OR { GREATER THAN / ≥ / LESS THAN / ≤ / EQUAL TO / = }] { VALUE n_2 / PRIORITY / PROXIMITY / m / -m }

Exemples

PRIORITY > 50
VALENCE AT ATOM 4 IS NOT LESS THAN OR EQUAL TO VALUE 1

Fonctionnement des enquêtes :

Dans les trois types d'enquêtes, on commence par évaluer la question de la façon suivante : on constate que dans tous les cas non numériques, la question se résume à une comparaison d'ensembles. On va donc déterminer <ensemble 1> et <ensemble 2> dans des set-registers internes, puis on fera entre eux l'opération demandée (AND, comparaison...). On obtient ainsi deux résultats :

- un ensemble résultat, qui est soit perdu, soit stocké dans le set register dont le N° est donné juste avant THEN ou en bout de ligne dans le type 3 (dans ce cas, le set register est obligatoire : une "enquête" de ce type ne sert qu'à le positionner).

- l'indication "vrai" ou "faux" , perdue dans le 3ème type, utilisée pour faire un choix dans les autres types :

. si on a "vrai", on exécute la phrase composée qui suit THEN, puis la 1ère phrase qui suit l'enquête, etc...

. si on a "faux", on exécute la 1ère phrase qui suit l'enquête dans le 1er type, et la phrase composée qui suit ELSE suivie de la 1ère phrase qui suit l'enquête dans le type 2.

Dans le cas des enquêtes numériques, le principe est le même, sauf que le type 3 est impossible et qu'on ne détermine pas d'ensembles, donc qu'on ne peut pas donner de numéro de set-register.

Définition d'une phrase composée : une phrase composée peut être :

- une directive seule
- une instruction seule
- la structure BEGIN
  suite de phrases quelconques y compris CLD et enquêtes
  DONE ↵

Exemple

```
IF ATOM IS ALPHA TO ATOM 5 THEN KILL
IF PRIORITY  >80 THEN CALL ARBUSOV AT ATOM 3
IF BOND 1-2 AND BOND 2-4 ARE CIS THEN
   BEGIN SUM 50 AND PRIORITY
         CALL ARBUSOV
   DONE
ELSE
   GO TO BOND124TRANS
```

Note :

la liste de phrases entre BEGIN et DONE peut être réduite à une seule phrase, en particulier à une enquête.

42B - les CLD

Lorsqu'on demande d'exécuter une directive unique, on la donne toute seule dans le programme de stratégie. Dès que l'on veut que s'exécutent plusieurs directives, elles devront obligatoirement être données sous forme de combinaison logique de directives (CLD). En effet, on doit dire si l'on veut qu'elles soient exécutées simultanément, ou si l'une des deux suffit,... La syntaxe d'une CLD est la suivante :

⊕ chaque directive est numérotée, ce numéro servant à un éventuel parenthésage ultérieur ; exemple :

1) BREAK BOND 5-6

⊕ deux directives qui se suivent sont supposées reliées entre elles par l'opérateur AND, qu'on n'écrit jamais explicitement :

4) REDUCE VALENCE AT 3
5) REDUCE VALENCE AT 5

} on veut réduire la valence des atomes 3 et 5.

⊕ les opérateurs OR et XOR doivent être écrits explicitement ; ils sont suivis de ↵ .

⊕ il n'y a pas de priorité implicite entre les opérateurs logiques.

⊕ on peut fermer une parenthèse en faisant suivre une directive (sur la même ligne ou sur la suivante) du caractère')' suivi du N° de la directive où se trouve la parenthèse ouvrante correspondante. Ce procédé permet de ne pas avoir à prévoir l'emplacement des parenthèses ouvrantes lors de l'écriture de la CLD.

⊕ on peut demander qu'une directive ou un groupe de directives incluses dans un parenthésage soit exécuté à l'exclusion de toute autre modification sur la molécule que celles qui sont explicitement demandées. Pour cela, faire suivre la directive ou la "parenthèse fermante" du mot ONLY.

Exemple :

```
MAKE ACID AT 5 WITH BONDS 3-4, 4-5, 4-8 ONLY
```

signifie qu'on veut créer un groupe ACID sur l'atome 5 à l'aide des liaisons 3-4, 4-5 et 4-8, mais qu'on ne veut aucune autre modification sur la molécule. Cette option peut être employée en ou hors CLD ; elle est à rapprocher de IGNORE, en permettant d'autres restrictions.

Exemple de CLD :

```
1) ADD + AT ATOM 5
2) INTRODUCE STEREO AT 1          )1
       OR
3) INTRODUCE ACID AT ATOM 3
     4) BREAK BOND 2-4
     5) MAKE RADICAL AT 2         )4
           OR
     6) REMOVE -AT 4              )4    )3
```

Si l'on remplace les directives par leur numéro, cette CLD équivaut à l'expression logique parenthésée suivante :

(1 AND 2) OR (3 AND ((4 AND 5) OR 6))

qui sera développée, puis exécutée en trois sous-objectifs :

1 AND 2
OR 3 AND 4 AND 5
OR 3 AND 6.

42C - Définition de choix, dans le cas d'un automate de stratégie

Il s'agit en quelque sorte d'un aiguillage permettant de définir plusieurs branches parallèles de programme qui doivent être essayées l'une après l'autre à partir du même précurseur (celui auquel on est arrivé avant de commencer la première branche). Deux cas sont à considérer :

* 1) choix hiérarchisé : le chimiste définit l'ordre dans lequel les branches s'exécutent. Si l'une s'avère impossible, on reprend avec la suivant, et ainsi de suite. Dès que le plan de stratégie proposé par l'une des branches d'un "choix hiérarchisé" a abouti, l'exécution de l'automate s'arrête. On construit ainsi un chemin unique.

* 2) choix non hiérarchisé : dans ce cas, aucun ordre ne peut être prédéfini entre les branches du choix, et toutes sont systématiquement essayées. L'automate construit ainsi autant de chemins issus du même point que de branches dont le plan de stratégie aboutit.

Il est possible d'imbriquer des "choix". Il n'est pas permis de pénétrer dans une branche d'un "choix" par un GO TO.

Si aucune des branches d'un choix ne permet d'aboutir, alors :

- si ce "choix" est défini à l'intérieur d'une branche d'un autre, on déclare cette branche impossible, et on essaye la suivante
- si l'on se trouve au niveau d'imbrication zéro, l'automate est déclaré impossible.

Syntaxe :

TRY [HIERARCHIC] CHOICE ↵
{ liste des phrases (quelconques) de la première branche
-NEXT CHOICE: ↵ } jusqu'à 7 fois
{ liste des phrases de la suivante }
ENDCHOICE
{ lorsqu'une branche est terminée avec succès, l'exécution continue à la

première phrase qui suit ENDCHOICE, à moins qu'un GO TO ne spécifie une autre adresse de suite.

42D - Début d'un programme de stratégie

On commence toujours par une ligne qui permet de donner un nom au programme et éventuellement de définir le motif sur lequel s'applique l'automate lorsque le programme décrit un automate.

Le nom du programme est une étiquette suivie du caractère ':' (définition d'étiquette normale), mais le reste de la ligne est soit vide, soit elle contient le motif, écrit dans le même format que les "patterns" d'ALCHEM. C'est à l'aide de ce nom de programme qu'on pourra faire référence à un automate depuis une autre stratégie.

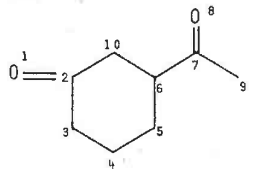
Description sommaire du motif : il s'agit d'une description linéaire d'une sous-structure, faites avec les conventions suivantes :

- les atomes sont représentés par leur symbole chimique (1 ou 2 lettres)
- les liaisons sont représentées par les caractères

- pour simple liaison
= pour double liaison
! pour triple liaison
? pour n'importe quelle liaison
% pour une liaison aromatique

- '@' suivi d'un numéro d'atome dans la liste indique la fermeture d'un cycle
- <liste des numéros d'atomes> indique une symétrie
- le codage de la sous-structure est terminé par le caractère '/'
- toute la suite de la carte est ignorée et peut contenir un commentaire
- des parenthèses servent à définir des chaînes qui partent toutes d'un même atome.

Exemple



s'écrit :

$$O_1 = C_2 - C_3 - C_4 - C_5 - C_6 (- C_7 ((= O_8) - C_9)) - C_{10} @ 2/$$

(@ 2 significant que le cycle se referme sur l'atome 2)

On trouvera plus de renseignements dans (13).

42E - Commentaires

Si une stratégie interactive, qui se bâtit au fur et à mesure en fonction des résultats affichés sur l'écran, nécessite peu (ou pas) de commentaires, il n'en est pas de même d'un automate, qui est conservé en bibliothèque et susceptible de servir plusieurs fois, longtemps après avoir été écrit. STRATOS permet donc l'emploi de commentaires : un commentaire commence avec le caractère ';' (comme dans ALCHEM) et se poursuit jusqu'en fin de ligne (colonne 72 sur carte). Le ';' peut être mis en premier caractère d'une ligne ; elle est alors entièrement prise en commentaire. Il peut aussi être placé après une phrase quelconque, le reste de la ligne sera pris en commentaire. L'apparition du signe ';' est équivalente à la rencontre de la fin de la ligne.

De plus, on peut introduire des lignes blanches n'importe où, elles sont transparentes.

42F - Fin d' un programme de stratégie

La dernière phrase est toujours constituée du seul mot END.

43 - AUTOMATES DE STRATEGIE ET STRATEGIE INTERACTIVE

On a déjà vu qu'en pratique, un automate de stratégie pourra être écrit à l'aide de toutes les phrases disponibles, mais avec très peu ou pas du tout de CLD, alors qu'une stratégie interactive employera principalement une CLD et très peu d'instructions.

En fait, on constate d'après ce qui précède que la structure donnée au langage ne fait aucune différence entre ces deux types de programme de stratégie (mis à part la donnée du motif). Aucune restriction n'est donc

imposée à l'utilisateur lorsqu'il a choisi son mode de travail.

44 - TRAITEMENT DES ERREURS

L'entrée des phrases se faisant souvent en conversationnel, le système de correction d'erreurs a été assez poussé. Nous allons voir ici principalement les moyens de correction mis à la disposition de l'utilisateur.

441 - Mode savant et mode débutant

Par une option donnée en début de compilation (voir § 45), on choisit entre ces deux modes.

Dans le mode savant, les messages-erreurs sont très brefs : ils désignent l'emplacement de l'erreur rencontrée et donnent le numéro du message explicite réservé au mode débutant.

Exemple

```
>ADD P AT ATOM 5
    ^
 CARACTERE OU MOT ERRONE
 ***CORRECTION ?
 ERR    1
>
```

Il est donc éventuellement possible de rechercher le message-explicite dans une liste si besoin est. De plus, on peut alors répondre :

```
   *HELP
```

ce qui provoque l'impression du message-explicite.

Dans le mode débutant, en plus des indications précédents, on obtient tout de suite le message explicite. *HELP est alors impossible.

Exemple

```
>MAKE +6M RING WITH ATOMS 1,2,3,4,8,9,10
     ^
 CARACTERE OU MOT ERRONE
 ***CORRECTION ?
```

```
 ERR 18
 SYNTAXE DE LA TAILLE D'UN CYCLE :
  6M+ OU >6M OU RM AVEC R=1,...,6
  DANS LES DEUX PREMIERS CAS (EQUIVALENTS), LA TAILLE EST LIMITEE A 10
  RETAPEZ OU  *MODIF
> {
```

442 - Correction automatique

STRATOS comporte un module de correction automatique, qui permet d'éviter à l'utilisateur de devoir corriger lui-même de simples erreurs de frappe ou d'orthographe du type

- lettre manquante ou en trop
- lettres interverties.

Si la correction automatique s'avère impossible, le système écrit simplement les messages normaux. Sinon, il signale la correction qu'il propose ; l'utilisateur répond par ↵ s'il est d'accord, sinon il envoie n'importe quelle correction "manuelle".

Exemple

```
>INTRODUCE STEERO AT 6
 MOT 'STEERO      ' REMPLACE PAR 'STEREO       '
 ***CORRECTION ?
>
>    {
```

443 - Réponses aux messages d'erreurs

Lorsque STRATOS signale une erreur, il rend la main à l'utilisateur pour correction, en écrivant

```
   ***CORRECTION ?
```

Tant que ce message apparaît, l'utilisateur peut donner d'autres corrections ; il répond par une ligne blanche (↵) pour indiquer qu'il a terminé ses corrections. Certaines corrections entraînent le retour immédiat dans l'analyseur.

Les phrases de correction sont les suivantes :

- retapez une nouvelle ligne : la nouvelle prend complètement la place de l'ancienne erronée et l'analyse repart tout de suite
- introduction d'un synonyme à l'aide de la phrase :

```
*SYNONYME nouveau-nom = ancien-nom
```

Le nouveau nom est introduit dans les tables du système et n'en sera pas oublié. Il est possible soit de donner d'autres corrections, soit de répondre ↵ pour refaire l'analyse de l'ancienne ligne inchangée compte tenu du nouveau synonyme.

Exemple (mode savant) :

```
>INTRODUCE ACIDE AT 3
            ^
 CARACTERE OU MOT ERRONE
 ***CORRECTION ?
 ERR   2
> * SYNONYME ACIDE=ACID
 ***CORRECTION ?
>
> {
```

- cas des erreurs sur l'ordre de grandeur d'un nombre : on retape simplement la nouvelle valeur. Si l'on veut complètement changer de ligne, répondre par * (seul sur la ligne), puis donner des corrections classiques.

Exemples (mode débutant) :

```
>IF BOND 23 IS AROMATIC THEN
          ^
 ERR 72
 DONNEZ UN NUMERO DE LIAISON ENTRE 1 ET 8
   RETAPEZ-LE
>6
> {
>1) ADD + AT 5
>ØR                ( ← zéro R au lieu de OR)
 ^
```

```
 ERR 105
 CE NUMERO A DEJA ETE EMPLOYE POUR UNE DIRECTIVE PRECEDENTE OU NUMERO HORS
  SEQUENCE
  RETAPEZ-LE OU * , PUIS CORRECTIONS
>*
 DONNEZ VOTRE NOUVELLE LIGNE, OU CORRECTIONS :
 OR
>2) {
    }
    } suite
```

- si le mot sur lequel est indiqué l'erreur en est la seule cause, introduire un message

*LIRE mot

Le mot donné remplacera simplement le mot qui a provoqué l'erreur. Cette possibilité n'est offerte que si l'erreur ne provient pas d'une autre partie de la phrase déjà analysée, mais du mot même qui a provoqué l'erreur

Exemples

```
>ADD * AT 5
     ^
 CARACTERE ERRONE
 ***CORRECTION ?
 LA SYNTAXE DEMANDE '+' OU '-'
>    * LIRE +
>  {
   }          le * est remplacé par +
>MAKE 3M RING WITH ATOMS 1,5,7,8
                          ^
 CARACTERE OU MOT ERRONE
 ***CORRECTION ?
 LE NOMBRE D'ATOMES OU DE LIAISONS SPECIFIE NE CORRESPOND PAS AVEC LA TAILLE
   INDIQUEE POUR LE CYCLE
   RETAPEZ LA LIGNE - *LIRE IMPOSSIBLE
```

En effet, l'erreur peut être sur le '3M' comme sur la liste d'atomes.

- si l'on est en mode savant, STRATOS écrit le N° du message-erreur.

La correction

*HELP

permet d'imprimer le message explicite correspondant.

Un 2ème appel au secours fait passer en mode débutant.

- le message

*COMMENT CORRIGER

imprime un résumé de ce paragraphe.

- modification d'une ligne sans la retaper : donner une correction

*MODIF , suivie de l'une des options suivantes :

⊕ a) un texte commençant en col 1 (éventuellement par des blancs) et limité à droite par le signe !

Ce texte sera inséré avant l'ancienne ligne :

```
>1) REMOVE ACID AT ATOM 1
>ADD + AT 5
   ^
 CARACTERE OU MOT ERRONE
 ***CORRECTION ?
 ERR 57
 APRES UNE DIR OU UNE CLD : ETIQUETTE, IF OU INSTRUCTION
 DANS LES AUTRES CAS  IDEM OU DIR OU NO DE DIR (CLD)
 DANS UN TEST : IDEM + ELSE
>* MODIF
 ***MODIF ?
 ADD + AT 5
>2)!
 ***MODIF ?
 2) ADD + AT 5
>
>  {
   }
```

⊕ b) insertion d'un texte au milieu de l'ancienne ligne : mettre le signe '+' sous le caractère précédent l'emplacement où il faut insérer, le faire suivre du texte à insérer, délimité à droite par !

```
>ADD AT 5
     ^
 CARACTERE OU MOT ERRONE
 ***CORRECTION ?
 ERR 1
 LA SYNTAXE DEMANDE '+' OU '-'
```

```
>*MODIF
 ***MODIF ?
 ADD AT 5
>   +- !
 ***MODIF ?
 ADD - AT 5
>
>
```

⊕ c) un signe '-' sous le 1er caractère d'un mot supprime ce mot :

```
>ADD ++ AT 5
      ^
 CARACTERE OU MOT ERRONE
 ***CORRECTION ?
 ERR 22
 LA SYNTAXE DEMANDE UN NUMERO D'ATOME
>*MODIF
 ***MODIF ?
 ADD ++ AT 5
>    -
 ***MODIF ?
 ADD + AT 5
>
>
```

⊕ d) un signe '=' sous le 1er caractère d'un mot, suivi d'un texte délimité à droite par ! , remplace ce mot par le texte :

```
>BREAK BOND 3-4, 4-5, 5-3
              ^
 CARACTERE OU MOT ERRONE
 ***CORRECTION ?
 ERR  24
 LE CYCLE A PLUS DE SOMMETS QUE VOUS EN AVEZ DECLARE
 OU : PLUS DE 10 SOMMETS
 OU : PLUS D'UN NUMERO (OU COUPLE) POUR DESIGNER UNE LIAISON
 RETAPEZ OU  *MODIF
>*MODIF
 ***MODIF ?
 BREAK BOND 3-4, 4-5, 5-3
>       =RING!
```



```
 ***MODIF ?
 BREAK RING 3-4, 4-5, 5-3
>
>
```

⊕ e) après chaque modif, le système imprime la ligne modifiée, puis est prêt à exécuter une nouvelle modification, jusqu'à ce qu'on lui envoie une ligne blanche (sortie de *MODIF).

⊕ f) pour avoir un résumé d'utilisation, taper *HELP après *MODIF .

⊕ g) il est possible de modifier ou de supprimer une partie d'un nom, depuis une lettre quelconque, jusqu'à la fin du nom ; il suffit pour cela de mettre le '=' ou le '-' sous le premier caractère à modifier et non pas sous le premier caractère du nom.

Exemple

```
>INTRODUCE STEREOCHIMIE AT ATOM 5
                    ^
 CARACTERE OU MOT ERRONE
 ***CORRECTION ?
 ERR   2
 LA SYNTAXE DEMANDE 'STEREOCHEMISTRY' OU UN GROUPE FONCTIONNEL
>*MODIF
 ***MODIF ?
 INTRODUCE STEREOCHIMIE AT ATOM 5
>                -
 ***MODIF ?
 INTRODUCE STEREO AT ATOM 5
>
>
```

On aurait aussi pu répondre :

```
 INTRODUCE STEREOCHIMIE AT ATOM 5
>                =CHEMISTRY!
 ***MODIF ?
 INTRODUCE STEREOCHEMISTRY AT ATOM 5
>
>
```

Exemple général :

```
>1) MAKE ACIDE AT ATON 3
 MOT 'ACIDE       ' REMPLACE PAR 'ACID      '
 ***CORRECTION ?
>*SYNONYME ACIDE=ACID
 ***CORRECTION ?
>
 MOT 'ATON        ' REMPLACE PAR ' ATOM      '
 ***CORRECTION ?
>
>1) MAKE ACIDE WITH BONS 1-4 2-3, 5-9
  ^
 NUMERO HORS SEQUENCE - RETAPEZ-LE
>2
 MOT 'BONS       ' REMPLACE PAR 'BOND        '
 ***CORRECTION ?
>*MODIF
 ***MODIF ?
 2) MAKE ACIDE WITH BONS  1-4 2-3, 5-9
>                  =RING !
 ***MODIF ?
 2) MAKE ACIDE WITH RING 1-4 2-3, 5-9
>
>  {
   }    suite
```

Dans cet exemple, on a compilé les phrases :

1) MAKE ACID AT ATOM 3
2) MAKE ACID WITH RING 1-4, 2-3, 5-9

45 - EMPLOI DU COMPILATEUR STRATOS

Lorsqu'on est en cours de recherche et d'essais de stratégie, il suffit de toucher le mot STRAT à l'aide du crayon lumineux pour provoquer son initialisation. Il répond en envoyant le message

STRATOS A VOTRE SERVICE

puis il demande les options de travail :

```
DONNEZ-MOI LES OPTIONS DE TRAVAIL :
 -MODE DEBUTANT ? (Y OU RETURN)
>réponse
 -LA STRATEGIE EST-ELLE DANS UN FICHIER ? (NOM DU FICHIER OU RETURN)
>
 -DOIT-ON CONSERVER LE CODE ? (NOM DU FICHIER OU RETURN)
>
 -DOIT-ON CONSERVER LE TEXTE SOURCE ? (NOM DU FICHIER OU RETURN)
>
VOTRE STRATEGIE, S'IL-VOUS-PLAIT :
```

STRATOS est alors prêt pour la compilation d'un programme de stratégie.

Il est également possible de faire la compilation d'un programme de stratégie hors du système de stratégie ; ce sera le cas des automates de stratégie, qui seront d'abord mis en bibliothèque avant de pouvoir servir. Pour cela, on appellera le compilateur à l'aide de l'ordre suivant :

@ STRATECOMP.STRATOS, options fichier source entrée, fichier source sortie, fichier code

Il y a deux types d'options :

- les options courantes : D = mode débutant
  S = mode savant
  C = entrée sur cartes ou fichier
  T = entrée au télétype
  L = impression d'un listing si l'entrée est sur cartes
  N = pas de listing
- les options maintenance : E = impression du code généré sous forme éclatée après chaque phrase
  F = impression du code généré compacté après chaque phrase
  G = impression du code généré complet après le END.

Si l'on ne donne aucune des options courantes, STRATOS les demandera explicitement en début de travail :

STRATOS A VOTRE SERVICE

DONNEZ-MOI LES OPTIONS DE TRAVAIL, SUR TROIS CARACTERES :
- MODE DEBUTANT OU SAVANT (D-S)
- CARTES OU TELETYPE (C-T)
- LISTING (L-N)
1 2 3
>réponse

Les options maintenance n'étant pas destinées à l'utilisateur chimiste, STRATOS ne les demande pas, dans ce cas.

Fichier source entrée

Donner le nom du fichier qui contient la source à compiler, si besoin est. Si l'entrée est sur cartes ou au télétype, ne rien mettre dans cette place.

Fichier source sortie

Nom du fichier qui contiendra la source après compilation. Cette option doit être utilisée dans le cas des automates en particulier, et dès qu'on veut employer l'option STRACE pendant la synthèse. On obtient dans ce fichier le texte source éventuellement corrigé.

Fichier code

Contient le code généré après compilation, avec les références au texte source si l'option fichier source sortie est présente. Doit être donnée pour les automates.

N.B. : même si l'un de ces noms de fichier est absent, il faut écrire les virgules.

Exemple

@ STRATECOMP.STRATOS, CLS , SOURCE, CODE

On travaillera en mode savant, sur cartes avec listing. La source sera mise dans le fichier SOURCE, et le code dans le fichier CODE.

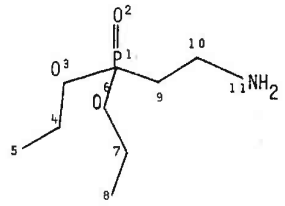
Notons que STRATOS se charge de l'assignation des fichiers demandée par le système EXEC 8 du 1110. L'utilisateur n'a pas besoin de s'en préoccuper.

Nous voyons donc que la définition de STRATOS permet autant que possible à l'utilisateur de s'exprimer à l'aide d'expressions de son langage

courant. De plus, il n'est pas prisonnier d'un carcan avec une syntaxe rigide. Enfin, il peut s'affranchir de connaissances concernant le système d'exploitation de l'ordinateur utilisé, puisque STRATOS peut prendre en charge lui-même les tâches d'assignation de fichier... et même d'édition de texte, tout au moins en ce qui concerne les modifications les plus courantes.

46 - EXEMPLE D'UTILISATION

A titre de test, nous avons soumis à un premier prototype de PASCOP travaillant avec STRATOS le problème de la synthèse d'un dérivé de l'acide amino-2 éthyl phosphonique (aussi appelé ciliatine ; c'est le premier composé à liaison P-C dont on a pu déceler la présence dans le règne animal, et sa synthèse a fait l'objet de nombreux travaux) :



Cette cible a été traitée une première fois par PASCOP sans faire appel à la stratégie, puis une seconde fois en utilisant les nouvelles possibilités offertes par STRATOS.

En fonctionnant sans objectif précis, PASCOP propose un premier niveau de six précurseurs (voir planche I), obtenus par la mise en oeuvre de transformations portant sur le groupe amino (NH_2), sans toutefois reconnaître l'importance stratégique de ces précurseurs. Ainsi, le précurseur (3) ouvre une première voie à la rupture rétrosynthétique de la liaison phosphore-carbone, qui constitue l'étape fondamentale de cette synthèse : cf. 2ème niveau.

Le temps relevé en fin d'exécution pour ces deux niveaux est de 47,609 secondes, dont 16,610 s.d'unité centrale et 16,421 s.d'entrées-sorties.

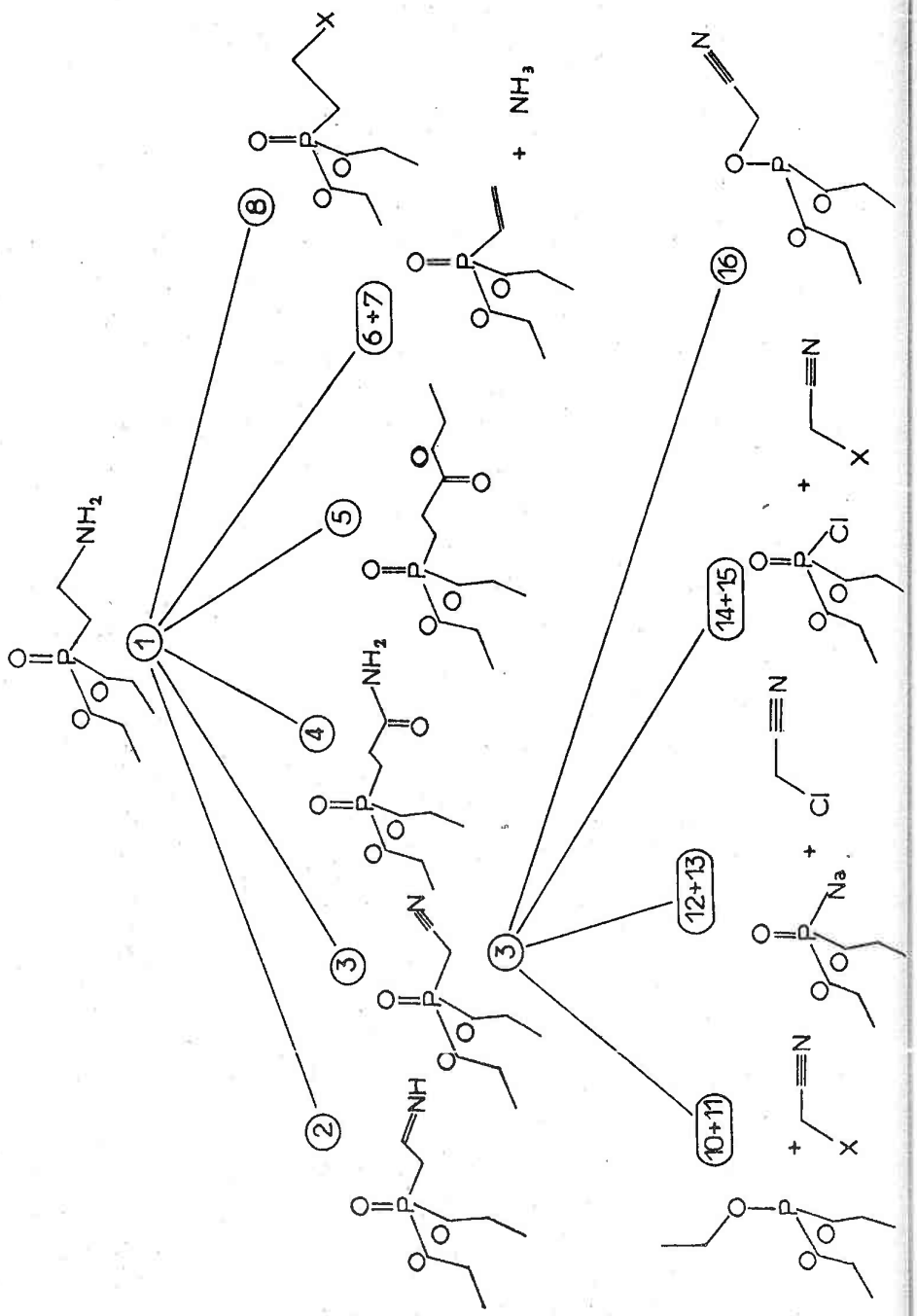


PLANCHE I : arbre de synthèse d'un dérivé de l'acide amino-2 éthyl phosphonique.

Par contre, grâce à STRATOS, le chimiste peut mettre en oeuvre une stratégie reposant sur la considération chimique suivante : les grandes réactions permettant de créer une liaison phosphore-carbone à partir d'un phosphite (Arbusov, Michaelis-Becker) ne pouvant s'appliquer directement dans le cas présent par suite de la présence dans la cible du groupe amino (NH_2), la stratégie pourrait reposer sur l'échange préalable du groupe amino, suivi d'un essai de rupture de la liaison P-C.

Pour cela, la création du premier niveau sera contrôlée par le programme de stratégie

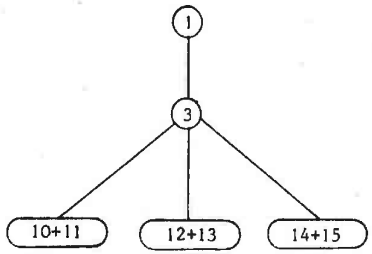
ETAPE1 :
EXCHANGE AMINE INTO NITRILE AT ATOM 11
END

On obtient ainsi uniquement le précurseur (3) .

Pour le second niveau, on proposera :

ETAPE2 :
BREAK BOND 1-9
END

A l'issue de ces opérations, la structure de l'arbre sera la suivante :



(numérotation de la planche I)

Temps d'exécution : 25,974 secondes, dont 6,719 s. d'unité centrale et 7,114 s. d'entrées/sorties.

PASCOP retrouve ainsi le chemin 10+11 → 3 → 1 correspondant à la synthèse expérimentale proposée par Isbell et coll. (34). Les deux autres voies, partant de 10+11 et de 14+15 n'ont pas d'équivalent expérimental, mais sont cependant également envisageables.

CONCLUSION

Le travail présenté dans cet ouvrage s'insérant dans le cadre d'une recherche pluridisciplinaire, sa conclusion va s'attacher à dégager les apports aux techniques informatiques d'une part, et au développement d'heuristiques nouvelles propres à la synthèse chimique assistée par ordinateur, d'autre part.

STRATOS a ouvert à PASCOP des champs d'applications indispensables à son futur développement. Il va permettre notamment de contrôler efficacement environ deux cents nouvelles transformations traitant les interchanges de groupes fonctionnels, transformations particulièrement importantes en synthèse organique, mais qui ne pouvaient être laissées à la libre disposition de PASCOP dans sa première version.

Par ailleurs, la stratégie interactive par directives va permettre d'accélérer d'une façon considérable la recherche de chemins de synthèse optimaux en dirigeant d'un point de vue chimique la création de l'arbre de synthèse. L'exemple comparatif donné en fin du chapitre IV donne une idée de ce fait. Cet aspect pourra encore être renforcé dans certains domaines de synthèses typiques par la mise en oeuvre d'automates qui permettent aux chimistes d'exploiter sur plusieurs étapes si nécessaire toutes les possibilités offertes par une transformation jugée particulièrement puissante sur le plan synthétique.

Une autre observation importante mérite d'être faite : généralement, les systèmes de synthèse assistée fonctionnant sans objectif précis n'entraînent pas d'emblée l'adhésion du chimiste de paillasse. L'une des principales causes du conflit vient du fait que l'utilisateur, consciemment ou non, s'attend à voir le système mettre en oeuvre des raccourcis ou des raisonnements relevant en fait largement de l'intuition ou de l'expérience, alors qu'un tel système procède par exploration rigide et complète de ses fichiers. Dans ce

domaine,un module de stratégie, tel que STRATOS, permettant à l'utilisateur de déployer des stratégies personnelles, devrait contribuer à réduire ces obstacles psychologiques.

Sur le plan pratique, il est encore difficile de juger l'approche choisie pour implanter toutes les nouvelles possibilités du système. En effet, toutes les améliorations prévues dans le cadre de la stratégie interactive ne sont pas encore opérationnelles : il est déjà possible de l'utiliser dans le cas "une étape", mais certains points sont encore en cours de test (cas "n étapes", cas ONLY et IGNORE), et d'autres attendent d'être programmés (XOR dans les CLD, correction automatique d'erreurs d'orthographe, minimalisation de la CLD si cela s'avère réellement utile). De plus, on ne peut pas encore essayer d'automates de stratégie, l'écriture des particularités dues à leur emploi n'étant pas entièrement terminée. Il est cependant d'ores et déjà possible de faire certaines constatations à partir des essais qui ont pu être faits.

On vérifie en particulier que la stratégie interactive permet réellement d'obtenir un arbre de synthèse débarassé des précurseurs-parasites que fournissait l'ancienne version du système. De plus, le gain de temps est appréciable, comme on l'a vu dans l'exemple comparatif. Enfin, il semble que les utilisateurs n'aient éprouvé aucune difficulté à apprendre à manier le langage STRATOS, d'autant plus que son emploi réserve beaucoup moins de surprises que celui d'ALCHEM, dont tous avaient déjà l'habitude.

En regardant PASCOP sous son aspect "système informatique", nous voyons en premier lieu que le module de stratégie qui y a été implanté a donné lieu à des études intéressantes sur des problèmes d'intelligence artificielle et de communication entre un système spécialisé et son interlocuteur non informaticien. En second lieu, il apparaît clairement qu'un tel système est loin d'être arrivé au terme de son développement. Et s'il peut encore bénéficier largement de résultats déjà obtenus dans le domaine de la reconnaissance des formes par exemple, il est certain qu'il devrait apporter sa contribution à l'avancement de la recherche sur l'intelligence artificielle ainsi que sur les problèmes de communication en langue naturelle.

Pour ce qui est de nouveaux développements à apporter à PASCOP, il semble que trois points principaux sont à constater. Tout d'abord, il faut permettre aux chimistes de travailler quelques temps sur la version obtenue

de PASCOP, pour qu'il soit possible de mieux déterminer ses défauts, mais aussi ses insuffisances. D'autre part, il est important de développer rapidement des moyens efficaces pour tenir compte de données économiques concernant les matières premières existantes en particulier ; en effet, un tel système ne trouvera normalement sa place dans l'industrie que s'il peut tenir compte de données aussi primordiales. Enfin, son "mode d'emploi" devra être aussi court et précis que possible, grâce à des techniques telles que celles qui ont été employées dans la définition de STRATOS. De grands progrès peuvent encore être faits dans ce domaine.

ANNEXES

ANNEXE A

TABLE DE LA GRAMMAIRE

CARLU = CARACTERE ENVOYE PAR L'ANALYSEUR LEXICAL
NEXTR = NUMERO DE LA REGLE SUIVANTE
SSAUTO = NUMERO DE SOUS-AUTOMATE
DERNR = INDICATEUR DE DERNIERE REGLE
FINAL = INDICATEUR D'ETAT FINAL
TACHE = NUMERO DE SSP TRAITANT LA SEMANTIQUE
NUMMES = NUMERO DE MESSAGE-ERREUR EXPLICITE

I	CARLU EN CLAIR	CODE	NEXTN	SSAUTO	DERNR	FINAL	TACHE	NUMMES	COMMENTAIRE
1	LABEL	001	2		*		25	59	NOM DU PROGRAMME DE STRATEGIE
2	:	023	3		*		21	55	
3	EOL	000	8				0	0	
4	DELTA	777	5		*		7	0	POUR LES AUTOMATES (COMPILE DANS TMOTIF)
5	EOL	000	8		*		0	56	FIN DU MOTIF DE L'AUTOMATE
6		776	0				0	0	
7	(*	776	93				0	0	RESERVE)
8	DO	003	24				0	0	DO NOT DIRECTIVE
9	VERBE-DIR	005	26				25	0	DIRECTIVE (COMPILEE DANS COMDIR)
10	NUMERO	021	30				10	0	PODIR1: INIT D'UNE CLD + PODIR
11	IF	006	51	121			3	0	DEPUT DE TEST
12	VERBE-INSTR	002	55				2	0	INSTRUCTION (COMPILEE DANS COMINS)
13	IGNORE	035	55				2	0	
14	STOP	032	104				23	0	SUITE DE 12: INSTRUCTION STOP
15	GO	033	102				0	0	GO TO
16	GOTO	034	103				0	0	GO TO
17	LABEL	001	22				20	0	LABEL DANS LE PROGRAMME
18	TRY	020	106				0	0	DEFINITION DES CHOIX
19	EOL	000	8				0	0	
20	END	015	0		*		17	57	FIN DU PROGRAMME DE STRATEGIE
21		776	0				0	0	
22	:	023	8		*		21	55	
23		776	0				0	0	
24	NOT	004	25		*		13	58	) DO NOT DIRECTIVE
25	VERBE-DIR	005	26		*		25	59	) (COMDIR)
26	ONLY	031	27				26	0	ONLY
27	EOL	000	11		*	*	0	60	
28		776	0				0	0	
29		776	0				0	0	
30	)	020	31		*		0	61	CLD: PARENTHESE OUVRANTE
31	DO	003	38				0	0	CLD
32	VERBE-DIR	005	34				1	0	CLD
33	IGNORE	035	34		*		1	62	CLD
34	ONLY	031	35				27	0	CLD
35	)	020	37				17	0	CLD)
36	EOL	000	40		*		17	63	CLD) FINDIR, PARENTHESE FERMANTE
37	NUMERO	021	50		*		15	64	CLD: NO DE PARENTHESE FERMANTE (PFDIR)
38	NOT	004	32		*		13	58	CLD
39		776	0				0	0	
40	NUMERO	021	30				12	0	CLD: AND IMPLICITE + PAR. OUVRANTE (PODIRE)
41	OR	013	48				14	0	CLD
42	XOR	014	49				15	0	CLD
43	)	020	37				0	0	CLD: PAR FERMANTE EN DEBUT DE LIGNE
44	DELTA	777	11		*	*	18	0	FIN DE CLD: AUTRE ORDRE (FINCLD)
45		776	0				0	0	
46		776	0				0	0	
47		776	0				0	0	
48	EOL	000	46				0	0	CLD)
49	NUMERO	021	30		*		11	66	CLD) PAR. OUVRANTE (PODIR)
50	ONLY	031	51				28	0	CLD

I	CARLU EN CLAIR	CODE	NEXTN	SSAUTO	DERNR	FINAL	TACHE	NUMMES	COMMENTAIRE
51	)	020	37				0	0	CLD: FIN DE PARENTHESE FERMANTE
52	OR	013	48				14	0	
53	XOR	014	49				15	0	
54	EOL	000	90		*		0	109	CLD
55	EOL	000	7		*	*	0	60	FIN D'INSTRUCTION, AU RETOUR DE COMINS
56		776	0				0	0	
57		776	0				0	0	
58		776	0				0	0	
59		776	0				0	0	
60		776	0				0	0	
61	THEN	027	63				5	0	TEST: TNTHEN = SAUT-SI-FAUX
62	EOL	000	7		*	*	24	67	TEST SANS THEN
63	BEGIN	011	74	8			7	0	TEST: BLOC THEN
64	DO	003	67				0	0	TEST)
65	VERBE-INSTR	002	55				2	0	TEST) ORDRE UNIQUE
66	VERBE-DIR	005	68		*		1	68	TEST)
67	NOT	004	66		*		13	58	TEST
68	ONLY	031	69				26	0	TEST
69	EOL	000	76		*		0	60	TEST: FIN D'ORDRE UNIQUE
70		776	0				0	0	
71		776	0				0	0	
72		776	0				0	0	
73		776	0				0	0	
74	DONE	012	89				8	0	TEST: FIN DE BLOC THEN
75	DELTA	777	74	8	*		0	0	TEST: SUITE D'UN BLOC
76	ELSE	010	77				4	0	TEST: THELSE = SAUT INCONDITIONNEL
77	DELTA	777	7		*	*	7	0	TEST SANS ELSE (FINIF)
78		776	0				0	0	TEST
79	BEGIN	011	80	8			7	0	TEST: BLOC ELSE
80	DO	003	83				0	0	TEST)
81	VERBE-INSTR	002	85				2	0	TEST) ORDRE UNIQUE
82	VERBE-DIR	005	84		*		1	68	TEST)
83	NOT	004	82		*		13	58	TEST
84	ONLY	031	85				26	0	TEST
85	EOL	000	7		*	*	7	60	TEST: FIN D'ORDRE UNIQUE
86		776	0				0	0	
87		776	0				0	0	
88		776	0				0	0	
89		776	0				0	0	
90	DONE	012	84				8	0	TEST: FIN DE BLOC ELSE
91	DELTA	777	90	8	*		0	0	TEST: FIN DE TEST AVEC ELSE (FINIF)
92		776	0				0	0	
93	(NUMERO	021	94	8	*		0	0	RESERVE)
94	(*	022	93				0	0	RESERVE)
95	(DELTA	777	8		*		0	0	RESERVE)
96		776	0				0	0	
97		776	0				0	0	
98		776	0				0	0	
99		776	0				0	0	
100		776	0				0	0	

I	CARLU EN CLAIR	CODE	NEXTR	SSAUTO	DERNR	FINAL	TACHE	NUMMES	COMMENTAIRE
101		775	0				0	0	
102	TO	017	103				0	0	GO TO
103	LABEL	001	104		*		22	55	GO TO
104	EOL	000	15		*	*	0	50	GO TO (APRES GO TO: LABEL POUR CONTINUER)
105		775	0				0	0	
106	HIERARCHIC	021	107				31	0	DEFINITION DES CHOIX
107	CHOICES	022	108		*		0	112	
108	EOL	000	105	8	*		30	112	
109	-	020	110				0	0	CHOIX SUIVANT
110	ENDCHOICE	024	8				33	0	FIN DE CHOIX
111	DELTA	777	109	8	*		0	113	
112	NEXT	023	113				0	0	
113	CHOICE	022	114				0	0	
114	:	023	115				0	0	
115	EOL	000	103		*		32	114	
116		775	0				0	0	
117		775	0				0	0	
118		775	0				0	0	
119		775	0				0	0	
120		775	0				0	0	********** ENQUETES **********
121	FAIL	155	0			*	99	0	CODOP
122	SUCCESS	154	0			*	99	0	=1
123	VALUE	150	445				100	0	)
124	PRIORITY	151	450				85	0	) TESTS
125	PROXIMITY	152	452				85	0	) NUMERIQUES, CODOP = 4
126	VALENCE	153	446				85	0	)
127	ELECTRO-	150	452				107	0	)
128	<PROPR1>	044	140				70	0	CATION, ANION, RADICAL)
129	STERIC	100	134				104	0	) TYPE 2
130	ELENERGY	101	147				104	0	) OU ENSEMBLE 1
131	NLENERGY	102	147				104	0	)
132	ATOM	103	150				58	0	) TYPE 5
133	(	026	156				0	0	) OU ENSEMBLE 1
134	DELTA	777	180	263	*		0	0	ENSEMBLE 1: TYPES 1, 3, 4 ET 5
135		775	0				0	0	
136		775	0				0	0	
137		775	0				0	0	
138		775	0				0	0	
139		775	0				0	0	
140	AT	104	253	253			42	0	) TYPE 2, VERS ENSEMBLE 1
141	ON	105	253	253			42	0	)
142	DELTA	777	180		*		0	0	ENSEMBLE 1 EN COURS
143		775	0				0	0	
144		775	0				0	0	
145		775	0				0	0	
146	HINDRANCE	106	147				0	0	
147	AT	104	253	253			0	0	
148	ON	105	253	253	*		0	84	
149		775	0				0	0	
150	NUMERO	021	152				55	0	<= 12

I	CARLU EN CLAIR	CODE	NEXTR	SSAUTO	DERNR	FINAL	TACHE	NUMMES	COMMENTAIRE
151	DELTA	777	180	281	*		0	0	ENSEMBLE 1 EN COURS
152	IS	107	154				102	0	TACHE 102: OP=2 + TACHE 54
153	DELTA	777	180	312	*		0	0	ENS 1 EN COURS
154		775	0				0	0	
155	NUMERO	021	156		*		54	85	NO DE SET-REGISTER <= 7
156	)	020	157		*		0	84	
157	IS	107	154				81	0	
158	DELTA	777	180		*		0	0	ENS 1 EN COURS
159		775	0				0	0	
160		775	0				0	0	
161		775	0				0	0	
162		775	0				0	0	
163		775	0				0	0	
164	NOT	004	165				43	0	
165	ATOM	103	173				58	0	
166	IN	111	174				0	0	TYPE 6
167	DELTA	777	180		*		0	0	ENS 1 EN COURS
168		775	0				0	0	
169		775	0				0	0	
170		775	0				0	0	
171		775	0				0	0	
172		775	0				0	0	
173	IN	111	175				0	0	)
174	DELTA	777	197		*		0	0	ENS 2 EN COURS (PAS TYPE 6)
175	POSITION	113	176		*		0	87	)
176	NUMERO	021	177		*		83	87	) TYPE 6
177	/	031	178		*		0	87	)
178	ATOM	103	179				0	0	)
179	NUMERO	021	508		*		84	87	) FIN DU TYPE 6
180	DELTA	777	181	382	*		0	0	FIN DE ENSEMBLE 1: RECHERCHE POSTKODIF EVENTUEL
181	IS	107	138				81	0	
182	AND	113	228		*		81	88	TYPE 3
183		775	0				0	0	
184		775	0				0	0	
185		775	0				0	0	
186		775	0				0	0	
187		775	0				0	0	
188	NOT	004	189				43	0	
189	ALL	119	197				44	0	TYPE 1
190	ATTACHED	116	210				0	0	TYPE 4
191	ON	105	217				0	0	TYPE 5
192	MOST	117	215				0	0	
193	LEAST	120	215				45	0	
194		775	0				0	0	
195		775	0				0	0	
196		775	0				0	0	
197	WITHIN	121	200				103	0	)
198	<POSIT>	043	201				51	0	)
199	DELTA	777	204	263	*		0	0	) ENS 2
200	<POSIT>	043	201		*		51	85	)

I	CARLO EN CLAIR	CODE	NEXTR	SSAUTO	DERNR	FINAL	TACHE	NUMMES	COMMENTAIRE
201	-	030	203				105	0	)
202	TO	017	209	253	*		105	50	) ENS 2
203	<POSIT>	043	202		*		51	85	)
204	DELTA	777	408	382	*		0	0	) FIN DES TYPES 1, 2 ET 5
205		776	0				0	0	
206		776	0				0	0	
207		776	0				0	0	
208		776	0				0	0	
209		776	0				0	0	
210	TO	017	211				0	0	)
211	ANY	123	213				50	0	) TYPE 4
212	NUMERO	021	213		*		51	51	) NO = 0 A 3
213	HYDROGENS	124	408		*		0	51	) FIN DU TYPE 4
214		776	0				0	0	
215		776	0				0	0	
216		776	0				0	0	
217	MOST	117	219				0	0	)
218	LEAST	120	219		*		45	52	)
219	HINDERED	126	220		*		104	52	) TYPE 5
220	SIDE	126	221		*		0	52	)
221	OF	127	197				0	0	)
222	DELTA	777	197		*		0	0	) VERS ENS 2
223		776	0				0	0	
224		776	0				0	0	
225		776	0				0	0	
226		776	0				0	0	
227		776	0				0	0	
228	WITHIN	121	231				103	0	)
229	<POSIT>	043	232				51	0	)
230	DELTA	777	235	253	*		0	0	)
231	<POSIT>	043	232		*		51	85	)
232	-	030	234				105	0	) TYPE 3
233	TO	017	235	253	*		106	50	)
234	<POSIT>	043	233		*		51	85	)
235	DELTA	777	241	382	*		0	0	)
236		776	0				0	0	
237		776	0				0	0	
238		776	0				0	0	
239		776	0				0	0	
240		776	0				0	0	
241	ARE	127	242		*		0	53	)
242	NOT	004	243				43	0	)
243	ALL	119	244				44	0	)
244	<TOPO>	041	408				104	0	) FIN DU TYPE 3
245	IN	111	246		*		0	53	)
246	SAME	130	247		*		104	53	)
247	RING	051	408		*		101	53	) FIN DU TYPE 3 (TACHE 101: EST-CE *RING*)
248		776	0				0	0	
249		776	0				0	0	
250		776	0				0	0	

I	CARLO EN CLAIR	CODE	NEXTR	SSAUTO	DERNR	FINAL	TACHE	NUMMES	COMMENTAIRE
251		776	0				0	0	
252		776	0				0	0	
253	DELTA	777	254	382	*		0	0	
254	NOT	004	255				43	0	)
255	BETTER	132	257				0	0	) TYPE 2
256	WORSE	133	257		*		45	84	)
257	THAN	134	197				81	0	)
258	DELTA	777	197		*		0	0	VERS ENS 2
259		776	0				0	0	
260		776	0				0	0	
261		776	0				0	0	-----ENSEMBLE----------
262	DELTA	777	0		*	*	0	0	RETOUR
263	ATOMS	103	281				58	0	)
264	BONDS	135	281				59	0	) CAS
265	GROUPS	136	270				0	0	) PARTICULIERS
266	PENTA	137	274				81	0	)
267	<SPREMOD>	042	262	321			52	0	AUTRES PREMODIFICATEURS
268	DELTA	777	262	321	*		0	0	PAS DE **
269		776	0				0	0	
270		776	0				0	0	
271		776	0				0	0	
272		776	0				0	0	
273		776	0				0	0	
274	-	030	277				0	0	
275	COORDINATED	140	262	321			0	0	
276	DELTA	777	262	321	*		0	0	
277	COORDINATED	140	262	321	*		0	84	
278		776	0				0	0	
279		776	0				0	0	
280		776	0				0	0	
281	WITHIN	121	284				103	0	
282	<POSIT>	043	285				51	0	
283	DELTA	777	262	301	*		0	0	
284	<POSIT>	043	285		*		51	85	
285	-	030	287				105	0	
286	TO	017	262	321	*		105	50	
287	<POSIT>	043	286		*		51	85	
288		776	0				0	0	
289		776	0				0	0	
290	WITHIN	121	284				103	0	
291	<POSIT>	043	285				51	0	
292	NUMERO	021	294				42	0	
293	DELTA	777	0		*	*	44	0	
294	ATOM	103	0			*	43	0	
295	BOND	135	0			*	44	0	
296	DELTA	777	0		*	*	45	0	
297		776	0				0	0	
298		776	0				0	0	
299		776	0				0	0	
300		776	0				0	0	

I	CARLU EN CLAIR	CODE	NEXTR	SSAUTO	DERNR	FINAL	TACHE	NUMMES	COMMENTAIRE
301	OTHER	191	307				0	0	) ATOM OTHER THAN
302	NUMERO	021	312				55	0	) ATOM N
303	DELTA	777	0		*	*	0	0	) ATOM
304		775	0				0	0	
305		775	0				0	0	
306		775	0				0	0	
307	THAN	134	308				0	0	)
308	NUMERO	021	309		*		57	55	)
309	-	030	311				0	0	)
310	DELTA	777	0		*	*	0	0	)
311	NUMERO	021	0		*	*	58	55	)
312	IN	111	314				0	0	) ATOM N IN GROUP M
313	DELTA	777	0		*	*	55	0	)
314	GROUP	136	315				0	0	)
315	NUMERO	021	0		*	*	57	55	) 1 OU 2
316		775	0				0	0	
317		775	0				0	0	
318		775	0				0	0	-----<OPERANDE>----------
319		775	0				0	0	
320		775	0				0	0	
321	<OP1>	050	0			*	53	0	EN 1 SEUL MOT
322	<OP1-2>	047	0			*	46	0	IDEM, SEULT VALABLES EN 2EME OPD
323	<OP2A>	045	330				41	0	SUIVIS DE ATOM OU RIEN
324	<OP2AB>	051	352				71	0	SUIVIS DE ATOM OU BOND OU RIEN
325	<OP2B>	052	357				41	0	SUIVIS DE BOND
326	<SDT>	045	355				41	0	SUIVIS DE BOND OU BOND ATOM (SIMPLE ...)
327	<PROPK1>	044	0			*	70	0	
328	HYDROGEN	129	0			*	0	0	)
329	ATOM	103	356	301			58	0	) CAS
330	BOND	135	355	301			55	0	) PARTICULIERS
331	GROUP	136	358				0	0	)
332	ACID	142	355				53	0	ACID OU ACID HALIDE
333	CARBONYL	143	357				53	0	CARBONYL GROUP
334	CARBONIUM	144	359				53	0	CARBONIUM ION
335	ALL	114	371				0	0	
336	STRONGLY	145	348				74	0	)
337	SLIGHTYL	146	348				77	0	) SEULEMENT
338	ELECTRO+	147	0			*	77	0	) EN 2EME OPD
339	ELECTRO-	150	0			*	77	0	)
340	ATOM	103	0			*	0	0	<OP2A> ATOM
341	(	024	377				0	0	SET REGISTER
342	DELTA	777	0		*	*	79	0	PAS D'OPERANDE, SEULT EN 2EME OPD
343		775	0				0	0	
344		775	0				0	0	
345		775	0				0	0	
346		775	0				0	0	
347		775	0				0	0	
348	ELECTRO+	147	0			*	78	0	
349	ELECTRO-	150	0		*	*	78	77	
350		775	0				0	0	

I	CARLU EN CLAIR	CODE	NEXTR	SSAUTO	DERNR	FINAL	TACHE	NUMMES	COMMENTAIRE
351	DELTA	777	0		*	*	0	0	<OP2A>
352	ATOM	103	0			*	72	0	<OP2AB> ATOM
353	BOND	135	0			*	73	0	<OP2AB> BOND
354	DELTA	777	0		*	*	74	0	<OP2AB>
355	BOND	135	353		*		0	12	<SDT> BOND
356	DELTA	777	0		*	*	0	0	
357	BOND	135	0		*	*	0	12	<OP2B> BOND
358	NUMERO	021	359				52	0	NO DE GROUPE: 1 OU 2
359	DELTA	777	0		*	*	55	0	GROUP
360	ATOMS	103	0			*	43	0	GROUP N ATOM
361	BONDS	135	0			*	54	0	GROUP N BOND
362	DELTA	777	0		*	*	55	0	GROUP N
363	ATOM	103	0			*	75	0	<SDT> BOND ATOM
364	DELTA	777	0		*	*	0	0	
365	HALIDE	151	0			*	57	0	
366	DELTA	777	0		*	*	0	0	
367	GROUP	136	0			*	0	0	
368	DELTA	777	0		*	*	0	0	
369	ION	152	0			*	0	0	
370	DELTA	777	0		*	*	0	0	
371	GROUP	136	0			*	56	78	
372		776	0				0	0	
373		775	0				0	0	
374		775	0				0	0	
375		775	0				0	0	
376		775	0				0	0	
377	NUMERO	021	378		*		54	85	SET REGISTER: <=7
378	)	020	0		*	*	0	86	
379		775	0				0	0	
380		775	0				0	0	
381		774	0				0	0	-----<POSTMODIFICATEUR>----FIN DE <ENSEMBLE>--
382	GAMING	153	382				48	0	
383	OFFRING	154	382				48	0	
384	GNPATH	155	0			*	47	0	
385	OFFPATH	156	0			*	47	0	
386	DELTA	777	0		*	*	0	0	
387		775	0				0	0	
388		774	0				0	0	
389		775	0				0	0	
390		776	0				0	0	
391		775	0				0	0	
392	OF	127	395				0	0	
393	SIZE	157	394				0	0	
394	DELTA	777	0		*	*	0	0	
395	SIZE	157	394		*		0	99	
396	OTHER	111	398				98	0	
397	NUMERO	021	400		*		99	99	
398	THAN	134	399				0	0	
399	NUMERO	021	400		*		99	99	
400	-	030	402				0	0	

I	CARLU EN CLAIR	CODE	NEXTR	SSAUTO	DERNR	FINAL	TACHE	NUMMES	COMMENTAIRE
%01	DELTA	777	0		*	*	0	0	
%02	NUMERO	021	0		*	*	50	99	
%03		776	0				0	0	-----FIN DE <POSTMODIF>---------
%04		776	0				0	0	
%05		776	0				0	0	
%06		776	0				0	0	
%07		776	0				0	0	
%08	(	024	%10				0	0	)
%09	DELTA	777	0		*	*	0	0	) FIN DES ENQUETES
%10	NUMERO	021	%11		*		82	85	) Y A-T-IL UN REGISTRE-RESULTAT
%11	)	020	0		*	*	0	85	)
%12		776	0				0	0	
%13		776	0				0	0	
%14		776	0				0	0	
%15		776	0				0	0	
%16		776	0				0	0	
%17		776	0				0	0	
%18		776	0				0	0	
%19		776	0				0	0	
%20		776	0				0	0	
%21		776	0				0	0	
%22		776	0				0	0	
%23		776	0				0	0	
%24		776	0				0	0	
%25		776	0				0	0	
%26		776	0				0	0	
%27		776	0				0	0	
%28		776	0				0	0	
%29		776	0				0	0	
%30		776	0				0	0	
%31		776	0				0	0	
%32		776	0				0	0	
%33		776	0				0	0	
%34		776	0				0	0	
%35		776	0				0	0	
%36		776	0				0	0	
%37		776	0				0	0	
%38		776	0				0	0	
%39		776	0				0	0	
%40		776	0				0	0	-----TESTS NUMERIQUES----------
%41		776	0				0	0	
%42		776	0				0	0	
%43		776	0				0	0	
%44		776	0				0	0	
%45	NUMERO	021	%58		*		85	100	
%46	AT	134	%50				0	0	)
%47	ATOM	103	%51				0	0	)
%48	NUMERO	021	%52				87	0	)
%49	(	024	%54		*		0	101	)
%50	ATOM	103	%51				0	0	)

I	CARLU EN CLAIR	CODE	NEXTR	SSAUTO	DERNR	FINAL	TACHE	NUMMES	COMMENTAIRE
%51	NUMERO	021	%52		*		87	101	) CAS VALENCE
%52	IN	111	%53				0	0	)
%53	DELTA	777	%58		*		0	0	)
%54	GROUP	136	%55				0	0	)
%55	NUMERO	021	%58		*		88	101	)
%56	NUMERO	021	%57		*		87	101	)
%57	)	020	%58		*		0	101	)
%58	IS	107	%59				0	0	
%59	NOT	004	%60				90	0	
%60	DELTA	777	%61	%72	*		0	0	
%61	OR	013	%66	%72			0	0	
%62	DELTA	777	%66		*		0	0	
%63		776	0				0	0	
%64		776	0				0	0	
%65		776	0				0	0	
%66	VALUE	160	%71				0	0	
%67	PRIORITY	161	0			*	91	0	) FIN DES
%68	PROXIMITY	162	0			*	91	0	) TESTS NUM
%69	-	030	%70				93	0	
%70	NUMERO	021	0		*	*	94	102	) FIN DES
%71	NUMERO	021	0		*	*	92	102	) TESTS NUM
%72	GREATER	155	%78				95	0	
%73	>	025	0			*	96	0	
%74	LESS	157	%78				95	0	
%75	<	026	0			*	97	0	
%76	EWUAL	170	%80				95	0	
%77	=	027	0		*	*	98	103	
%78	THAN	134	0			*	0	0	
%79	DELTA	777	0		*	*	0	0	
%80	TO	017	0			*	0	0	
%81	DELTA	777	0		*	*	0	0	
%82		776	0				0	0	
%83		776	0				0	0	
%84		776	0				0	0	
%85		776	0				0	0	
%86		776	0				0	0	
%87		776	0				0	0	
%88		776	0				0	0	
%89		776	0				0	0	
%90		776	0				0	0	
%91		776	0				0	0	-----FIN DES TESTS NUMERIQUES----------
%92		776	0				0	0	
%93		776	0				0	0	
%94		776	0				0	0	
%95		776	0				0	0	
%96		776	0				0	0	
%97		776	0				0	0	
%98		776	0				0	0	
%99		776	0				0	0	
%00		776	0				0	0	

FIN

ANNEXE B

A titre d'exemple sur les automates de stratégie, cette annexe donne une version simplifiée de l'automate écrit par M.H. ZIMMER, permettant la création de la liaison P-C du motif P(=O)—C, par des réactions spécialisées, celles d'Arbusov ou de Michaelis-Becker étant les plus importantes. Dans le sens rétrosynthétique, cela revient en fait à casser la liaison P-C.

* 1) Motif de départ : P(=O)—C

* 2) Dans un "programme principal", on teste la présence effective de ce motif, ainsi que son environnement dans la cible, entre autres :

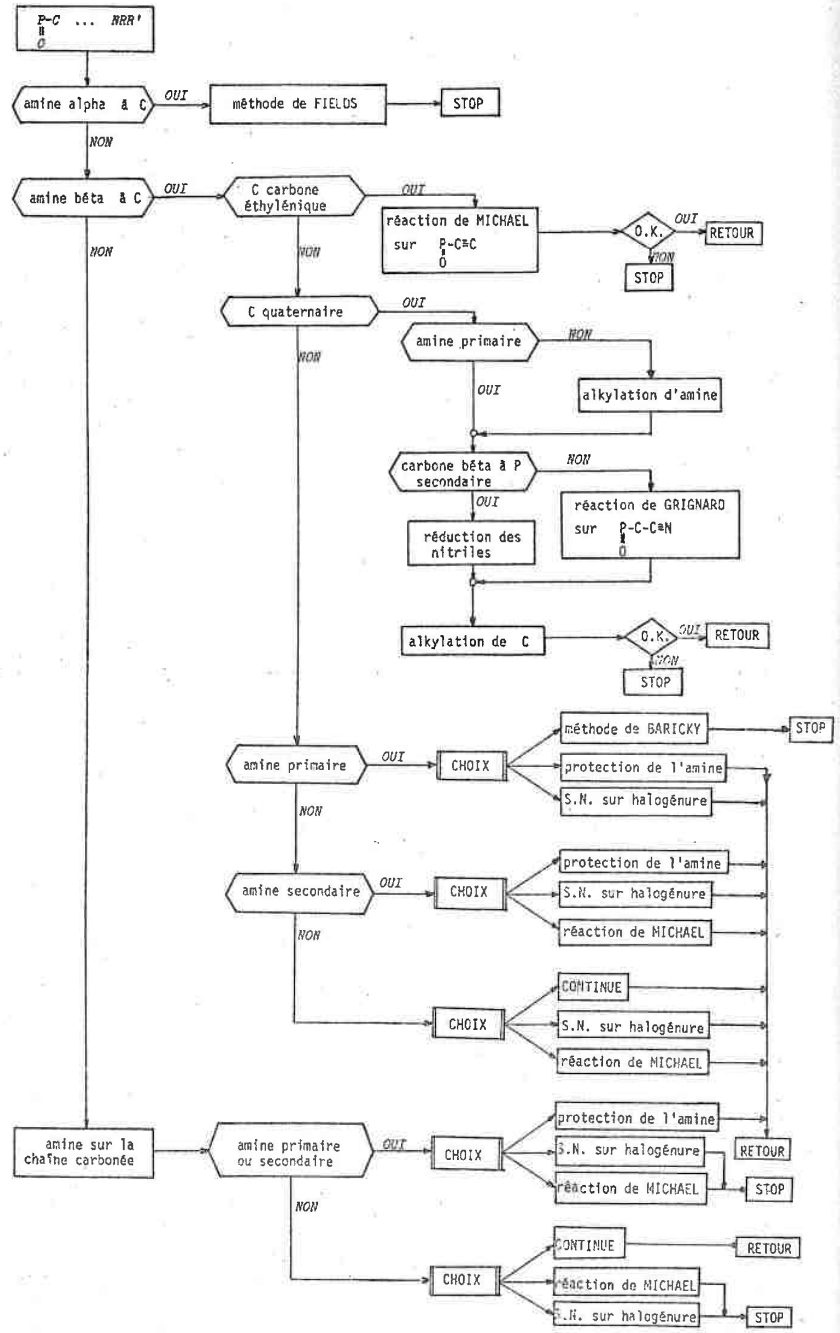
a - la liaison P-C est-elle une liaison cyclique ?
b - l'atome de carbone est-il quaternaire (quatre atomes autres que des atomes d'hydrogène lui sont rattachés) ?
c - l'atome en alpha du phosphore est-il un atome d'hydrogène ?
d - l'atome en alpha du phosphore est-il un hétéroatome (par exemple un atome de chlore, d'oxygène ou d'azote) ?
e - l'atome de carbone fait-il partie d'un cycle ?
f - y a-t-il une amine sur la chaîne carbonée ?
g - l'atome de carbone est-il un carbone éthylénique ?
h - l'atome de carbone est-il en alpha d'une cétone ou d'un aldéhyde ?
i - etc...

En réponse à chacune de ces questions, on définit un ou plusieurs objectifs intermédiaires destinés à préparer la cible à l'application de la transformation d'Arbusov ou de Michaelis-Becker ; en cas d'echec, on essaie d'appliquer une autre transformation.

* 3) Nous allons étudier plus précisément la création des objectifs intermédiaires dans le cas "f" (amine sur la chaîne carbonée), à l'aide de l'organigramme ci-après.

* 4) Traduction de cet organigramme dans le langage STRATOS :

Partie de l'organigramme de l'automate ARBUZOV traitant les aminoalkylphosphonates ou -phosphinates. 172




```
ARBUSOV:        P(=O)-C/
      {
;
; TRAITEMENT DES AMINOALKYLPHOSPHONATES OU -PHOSPHONITES
;
IF ATOM 3 IS AMINE THEN
    BEGIN
    CALL FIELDS
    STOP
    DONE
IF AMINE IS NOT ALPHA TO ATOM 3 OFFPATH THEN GO TO SUPALPHA
IF ATOM 3 IS OLEFIN THEN            ; TEST 'C CARBONE ETHYLENIQUE'
    BEGIN
    IF ATOM IS ALPHA TO ATOM 3 OFFPATH (1) ; DEFINITION
    IF (1) IS AMINE (2)                    ;            D'ENSEMBLES
    CALL MIK/AMINE AT (2)
    IF FAIL THEN STOP
    ELSE GO TO RETOUR
    DONE
IF ATOM 3 IS QUATERNARY THEN
    BEGIN
    IF AMINE ALPHA TO ATOM 3 OFFPATH  IS NOT PRIMARY THEN
         BEGIN
         CALL ALKYLAMINE
         IF FAIL THEN STOP
         DONE
    IF (2) IS SECONDARY THEN
         CALL 10AMINE
    ELSE
         CALL 6+NITRILE AT (2)
    CALL ALKYLATION AT ATOM 3
    IF FAIL THEN STOP
    ELSE GO TO RETOUR
    DONE
IF AMINE ALPHA TO ATOM 3 OFFPATH IS PRIMARY THEN
    BEGIN
```

```
    TRY HIERARCHIC CHOICE
          CALL BARICKY
          STOP
    - NEXT CHOICE :
          CALL PROPTECTAMINE
          GO TO RETOUR
    - NEXT :
          CALL 4HALOGENURE AT (2)
          GO TO RETOUR
    ENDCHOICE
    DONE
ELSE
    BEGIN
    TRY CHOICES
          IF AMINE ALPHA TO ATOM 3 IS SECONDARY THEN CALL PROTECTAMINE
          GO TO RETOUR
    - NEXT CHOICE:
          CALL 4HALOGENURE AT (2)
          GO TO RETOUR
    - NEXT CHOICE:
          CALL MIK/AMINE AT (2)
          GO TO RETOUR
    ENDCHOICE
DONE
;
SUPALPHA:
IF AMINE IS ANYWHERE (3)
TRY HIERARCHIC CHOICES
    IF (3) IS TERTIARY AMINE THEN GO TO RETOUR
    ELSE
          BEGIN
          CALL PROTECTAMINE
          GO TO RETOUR
          DONE
- NEXT:
    IF (3) IS TERTIARY AMINE THEN
         BEGIN
         CALL MIK/AMINE
```



```
          STOP
          DONE
    ELSE
          BEGIN
          CALL 4HALOGENURE
          STOP
          DONE
- NEXT:
    IF (3) IS TERTIARY AMINE THEN
          BEGIN
          CALL 4HALOGENURE
          STOP
          DONE
    ELSE
          BEGIN
          CALL MIK/AMINE
          STOP
          DONE
;
RETOUR:
```

ANNEXE C

1) Caractères utilisés dans les transformations :

N° de la chaîne de bits	CHARACTERS	N° de la chaîne de bits	CHARACTERS
1	BREAKS CC BOND	29	MAKES 4M RING
2	" CN "	30	" 5M "
3	" CP "	31	" 6M "
4	" CO "	32	" 6M+ "
5	" CS "	33	" SUBSTITUTION
6	" CX "	34	" RADICAL
7	" NO "	35	" groupe
8	" C-HETERO BOND	36	REDUCES CC BOND
9	" NN BOND	37	" CN "
10	" BOND	38	" NN "
11	" RING	39	" RING
12	" 3M RING	40	" COORDINENCE
13	" 4M "	41	" VALENCE
14	" 5M "	42	INCREASES CC BOND
15	" 6M "	43	" CN "
16	" 6M+ "	44	" NN "
17	MAKES CC BOND	45	" RING
18	" CN "	46	" COORDINENCE
19	" CP "	47	" VALENCE
20	" CO "	48	AROMATIZES CC BOND
21	" CS "	49	DEAROMATIZES CC BOND
22	" CX "	50	INVERTS SP2 ATOMS
23	" NO "	51	" SP3 ATOMS
24	" C-HETERO BOND	52	" A DOUBLE BOND
25	" HETERO-HETERO BOND	53	MODIFIES ATOM STRUCT TO TBP
26	" BOND	54	" " " TO SBP
27	" RING	55	" " " TO OCTAHEDRAL
28	" 3M RING		

N° de la chaîne de bits	CHARACTERS
56	MODIFIES ATOM STRUCT TO SP3
57	" COORDINENCE
58	LOSES STEREOCHEMISTRY
59	INTRODUCES STEREOCHEMISTRY
60	" groupe
61	EXCHANGES groupe1/groupe2
62	REMOVES groupe
63	" CHARGE +
64	" CHARGE -
65	ADDS CHARGE +
66	ADDS CHARGE -
67	REARRANGES
68	TYPE a/b
69 à 100	réservés pour usage ultérieur

2) Quelques statistiques montrant l'emploi des divers caractères, calculées sur l'état actuel du fichier:

- actuellement, 68 caractères sont prévus, mais seulement 48 servent effectivement
- 7 caractères sont utilisés dans plus de 10 % des transformations :

EXCHANGES groupe1/groupe2	41,6 %	} mais ces deux sont en
MAKES groupe	37,7 %	} fait décomposés
BREAKS CC BOND	35,5 %	
REDUCES COORDINENCE	30,4 %	
INCREASES CC BOND	14,2 %	
MODIFIES COORDINENCE	12,8 %	
REDUCES CC BOND	10,6 %	

- 6 ne sont utilisés que dans 5 à 10 % des transformations
- 18 dans 1 à 5 %
- 15 dans moins de 1 %.

On constate donc que la sélection des transformations par les caractères a de bonnes chances d'être efficace, sauf pour BREAKS CC BOND et REDUCES COORDINENCE qui retiennent encore environ un tiers du fichier.

- pourcentage maximal de transformations pouvant faire au moins une action prise parmi p :

p = 1	35,5 %	p = 5	59,5 %
p = 2	45,5 %	p = 6	64,7 %
p = 3	50 %	p = 7	65,4 %
p = 4	56,4 %		

On constate donc que l'efficacité de cette sélection diminue assez vite lorsque le nombre de caractères demandés augmente. Mais il faut être conscient de la valeur approximative de ces chiffres.En effet, ils ont été calculés d'après les sept caractères les plus employés pour qu'on obtienne les valeurs maximales. Mais il faut se souvenir en particulier que les deux caractères en fait les plus employés (EXCHANGES groupe1/groupe2 et MAKES groupe), une fois précisés par le(s) groupe(s), ne concerneront plus que une à trente transformations, donc moins de 3 % (voir § 2214, partie sur chaînes de bits spéciales).

- enfin, notons que les directives les plus employées ne seront peut-être pas celles qui correspondent aux caractères les plus employés, mais ceci est difficile à chiffrer car dépendant du sujet étudié, de l'utilisateur, etc...

BIBLIOGRAPHIE

1. a) E.J. COREY, W.T. WIPKE
Science, 166, 178 (1969)

b) E.J. COREY , W.T. WIPKE, R.D. CRAMER, W.J. HOWE
J. Am. Chem. Soc., 94, 421 (1972)

c) E.J. COREY, W.T. WIPKE, R.D. CRAMER, W.J. HOWE
J. Am. Chem. Soc., 94, 431 (1972)

d) E.J. COREY, R.D. CRAMER III, W.J. HOWE
J. Am. Chem. Soc., 94, 440 (1972)

2. W.T. WIPKE
Computer Representation and Manipulation of Chemical Information,
Wiley Interscience, N.Y. 1974, p. 147

3. a) I. UGI, P. GILLESPIE
Angew. Chem., 83, 980 (1971)
Angew. Chem., Inter. Edit. 10, 914 (1971)

b) I. UGI, P. GILLESPIE
Angew. Chem., 83, 982 (1971)
Angew. Chem., Intern. Edit., 10, 915 (1971)

4. H. GELERNTER, N.S. SRIDHARAN, A.J. HART, S.C. YEN, F. FOWLER et H.SHUE
Top. Curr. Chem., 41, 113 (1973)

5. a) R. BARONE, M. CHANON, J. METZGER
Rev. Inst. Fr. Pēt. Ann. Combust. Liq., 28, 771 (1973)

b) R. BARONE, M. CHANON, J. METZGER
Tetrahedron Lett., 1974, p. 2761

6. M. BERSOHN
Bull. Jpn. Chem. Soc., 45, 1897 (1972)

7. E.J. COREY
Pure Appl. Chem., 14, 19 (1967)

8. E.J. COREY, W.J. HOWE, D.A. PENSAK
J. Am. Chem. Soc., 96, 7724 (1974)

9. DARC System : System of Description, Acquisition, Retrieval and Conception.
a) Structural organic thinking and computer assistance in synthesis and correlation (Israël Journal of Chemistry, vol. 14, 1975, p. 17 à 32)
b) Application des traitements graphiques en documentation automatique et en conception assistée en chimie (conférence prononcée lors des Journées Graphiques AFCET-IRIA ; 5-7 décembre 1973).

10. T.M. DYOTT
Programme "Simulation and Evaluation of Chemical Synthesis", Ph. D. Thesis, Princeton University, 1973

11. a) Logiciel VERONICA
Centre Régional de Calcul de l'Université de Copenhague (Danemark, RECKU, publication n° 10, 1972)
b) F. SABOURIN, G. KAUFMANN
Version UNIVAC 1110 du logiciel Veronica, Laboratoire de Spectrochimie Moléculaire, rapport interne, 1975

12. R. MARC : contribution au développement d'un système informatique d'aide à l'élaboration de chemins de synthèse des composés organophosphorés (thèse de docteur-ingénieur, 1977)

13. W.T. WIPKE, W.J. HOWE : computer-assisted organic synthesis - p.105 (ACS symposium series, 61)

14. W.T. WIPKE, T.M. DYOTT, C. STILL et P. FRIEDLAND, non publié

15. ALDRICH-EUROPE : Catalog/Handbook of Organic and Biochemicals (Janssen Pharmaceutica - 1976)

16. J.C. DERNIAME, C. PAIR : problèmes de cheminement dans les graphes (AFCET - Dunod 1971)

17. J.C. GRATTAROLA : conception et réalisation de fichiers en temps réel (thèse de spécialité informatique - 1976)

18. D. GRIES : compiler construction for digital computers (John Wiley & sons, inc. 1971) : chapitre 12.

19. Comme 18 - paragraphe 9.3

20. Comme 18 - chapitre 21, p. 451

21. P.PFAADT : DEA d'informatique/systèmes - 1977

22. C. PAIR : Informatique non numérique - structures de données et algorithmes fondamentaux (Institut National Polytechnique de Nancy)

23. C. CORAY, G. STAMMON : éléments de reconnaissance des formes (Ecole Internationale d'été d'Informatique AFCET - 1974)

24. R.S. LEDLEY, J.B. WILSON : programming and utilizing digital computers (Mc Graw - Hill book company inc).

25. H.L. MORGAN
J. Chem. Doc., 5, 107 (1965)

26. F. CHOPLIN, R. MARC, C. LAURENÇO, G. KAUFMANN, W.T. WIPKE : Nouveau Journal de Chimie (sous presse)

27. H.L. MORGAN : spelling corrections in system programs (communication of ACM,volume 13, février 1970, p. 90-94)

28. F.J. DAMERAU : a technique for computer detection and correction of spelling errors (communication of ACM, volume 7, mars 1964)

29. A.U. AHO, J.D. ULLMANN : the theory of parsing, translation and compiling - volume 1 : parsing - p.112

30. R.W. LUCKY, J. SALZ, E.J. WELDON : principles of data communication (Mc Graw - Hill book company - p. 284)

31. J. KUNTZMANN : algèbre de Bool (Dunod 1965)

32. a) E.J. McKLUSKEY : minimization of boolean functions (Bell system techn. J, vol. 35, p. 1417, novembre 1956)
b) W.O. QUINE : the problem of simplifying truth functions (AM. Math., monthly, vol. 59 N° 8, p. 521-531, octobre 1952)

33. B. HARRIS : an algorithme for determining minimal representation of a logical function (IRE trans. on electronic computers, vol. EC-6, p.103-108, juin 1957)

34. A.F. ISBELL, J.P. BERRY, L.W. TANSEY, J. Org. Chem., 37, 4399 (1972).

NOM DE L'ETUDIANT : M. BONNET PATRICK

NATURE DE LA THESE : SPECIALITE INFORMATIQUE

VU, APPROUVE

et PERMIS D'IMPRIMER

NANCY LE 22 FEV 1978 1000

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

UNIVERSITE NANCY I LE PRESIDENT

M. BOULANGE