INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

Département de Chimie Physique des Réactions

Centre de Recherche en Informatique de Nancy

Mise en oeuvre informatique de la modélisation de réactions chimiques

Thèse soutenue le 1er octobre 1982 par

LAURENCE HAUX épouse VOGIN

pour obtenir le titre de

Docte[illegible] informatique

devan[illegible] :

Président : Monsieur J.C. DERNIAME

Examinateurs : Messieurs G.M. CÔME
D. COULON
P.Y. CUNIN
M. GRIFFITHS
F. JAKOBIAK
M. NICLAUSE

INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

Département de Chimie Physique des Réactions

Centre de Recherche en Informatique de Nancy

Mise en oeuvre informatique de la modélisation de réactions chimiques

Thèse soutenue le 1er octobre 1982 par

LAURENCE HAUX épouse VOGIN

pour obtenir le titre de
Docteur-Ingénieur spécialité informatique
devant la commission d'examen :

Président : Monsieur J.C. DERNIAME

Examinateurs : Messieurs G.M. CÔME
D. COULON
P.Y. CUNIN
M. GRIFFITHS
F. JAKOBIAK
M. NICLAUSE

Je tiens à remercier ici

Monsieur DERNIAME qui a accepté de présider le jury d'examen.

G.M. CÔME et M. GRIFFITHS, mes directeurs de recherche. Ils sont, avec P.Y. CUNIN, les maîtres d'oeuvre de la collaboration entre chimistes et informaticiens ; c'est à eux que je dois mon sujet de thèse qu'ils m'ont laissé développer librement, tout en restant très disponibles pour m'aider à résoudre mes problèmes, et je suis très sensible à la confiance qu'ils m'ont accordée.

Monsieur NICLAUSE de m'avoir accueillie dans son laboratoire et d'avoir bien voulu participer à ce jury.

D. COULON qui, depuis le départ de la première équipe, assure la poursuite des recherches en informatique chimique à Nancy.

Du côté industriel, je remercie les personnes de P.C.U.K. : Monsieur JAKOBIAK qui amène au jury son expérience de l'informatique chimique, et Monsieur WEISS qui, en tant que chimiste, a accepté de me parrainer auprès du C.N.R.S. .

Enfin je remercie tous ceux et celles qui m'ont aidée au cours de ce travail et qui ont participé à la réalisation matérielle de cette thèse.

S O M M A I R E

I N T R O D U C T I O N

En sortant de l'E.N.S.E.E.I.H.T. (Ecole Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique et d'Hydraulique de Toulouse) en 1979, j'ai obtenu mon diplôme d'ingénieur en informatique. L'acquisition d'une bourse C.N.R.S. m'a amenée au Département de Chimie Physique des Réactions (L.A. 328 du C.N.R.S.), en vue de la préparation d'une thèse de docteur-ingénieur.

Cette thèse fait partie d'un projet d'informatique appliqué à la chimie, né à Nancy en 1977, grâce à la collaboration entre le laboratoire de Cinétique Chimique Appliquée du L.A. 328 du C.N.R.S. et l'équipe d'Ingéniérie du Logiciel du Centre de Recherche en Informatique de Nancy (L.A. 262 du C.N.R.S.).

=:=:=:=:=

En chimie comme dans bien d'autres disciplines les spécialistes éprouvent le besoin de disposer d'outils informatiques, dont la mise en oeuvre fait appel à des techniques appartenant à des domaines variés : traduction de langage, intelligence artificielle, bases de données, théorie des graphes, traitements graphiques, etc.., Les projets d'informatique chimique les plus importants concernent, par exemple, la synthèse organique assistée par ordinateur, les bases de données de molécules, l'élucidation de structures moléculaires à partir de données spectroscopiques, la corrélation entre activité et structure.

La cinétique chimique n'échappe pas à ce besoin d'outils, car la description de réactions chimiques complexes nécessite des mécanismes réactionnels comportant quelques dizaines de constituants et quelques centaines d'équations chimiques appelées processus élémentaires. On comprend donc

l'intérêt de pouvoir analyser toutes ces équations dans un contexte informatique.

Une première étape dans l'informatisation de l'étude des mécanismes réactionnels a été la mise en oeuvre d'un compilateur de langage chimique. L'idée originale de ce travail est de manipuler des formules chimiques comme un langage de programmation. La syntaxe du langage a été définie par G.M. CÔME, P.Y. CUNIN, M. GRIFFITHS et C. PAIR. Une première version du compilateur a été réalisée par S. LAFONT [LAFONT 1977], et une deuxième par D. ALRAN [ALRAN 1979].

La deuxième étape est amorcée par le travail présenté : alors que le compilateur traite les données chimiques essentiellement au niveau syntaxique (l'utilisateur doit entrer lui-même les équations chimiques du mécanisme réactionnel à étudier), nous avons envisagé d'écrire automatiquement les processus élémentaires d'une réaction à partir d'une liste de réactifs.

Après avoir situé le contexte chimique du projet (chapitre I), nous analysons le problème de la construction automatique de mécanismes réactionnels (chapitre II). Le chapitre III est consacré à la définition d'une notation chimique externe et au choix d'une représentation interne des constituants. Dans le chapitre IV, nous détaillons les algorithmes utilisés pour créer les processus élémentaires. Enfin dans le chapitre V, nous présentons quelques aspects propres à une implantation du projet en langage PASCAL.

CHAPITRE I

Le contexte physico-chimique du projet

Il apparaît utile dans un premier temps, pour situer notre projet dans son contexte physico-chimique, de donner quelques indications générales sur les buts poursuivis par la cinétique chimique, la méthodologie qu'elle utilise pour les atteindre, les concepts et les modèles qui lui permettent de rendre compte ou de prédire des phénomènes cinétiques expérimentaux.

Nous pourrons alors procéder à la description des modèles cinétiques fondamentaux que sont les mécanismes réactionnels.

Nous terminerons ce chapitre en examinant les problèmes posés par le traitement de mécanismes réactionnels complexes à l'aide de compilateurs spécialisés.

1. LA CINETIQUE CHIMIQUE.

G.M. CÔME et M. NICLAUSE [CÔME 1980, 1981] ont défini de la manière suivante la cinétique chimique et la méthodologie correspondante :

"La première question qui se pose dans l'étude des réactions chimiques est de savoir si, dans certaines conditions, un système donné est susceptible d'évoluer d'un état initial I vers un certain état final F. A cette question, la thermodynamique chimique permet, en principe, de répondre soit par *non*, soit par *peut-être*. Autrement dit, la thermodynamique chimique permet généralement de savoir si une réaction est (thermodynamiquement) possible.

Mais une telle réaction n'est pas effective pour autant : le système peut rester quasi-indéfiniment en état de *faux équilibre*, ou évoluer vers un autre état final F' par une réaction, (elle aussi thermodynamiquement possible) plus rapide que la réaction attendue. Le facteur *vitesse* échappe donc à la thermodynamique. C'est une lacune que se propose de combler la *cinétique chimique*, qui s'intéresse donc tout d'abord à la *vitesse* des transformations chimiques, et qui étudie systématiquement les différents facteurs qui influent sur cette vitesse. Mais la loi de vitesse est une conséquence de son *mécanisme*, c'est-à-dire de la façon dont se déroule la réaction à l'échelle moléculaire. Il en résulte que l'étude de la vitesse est, en fait, intimement liée à l'analyse du mécanisme de réaction, de sorte que la cinétique chimique désigne la discipline physico-chimique moderne qui étudie, à la fois, la vitesse et le mécanisme des réactions chimiques.

L'étude cinétique d'une réaction chimique commence (après une réflexion théorique préliminaire) par une phase *d'expérimentation*, consistant à collecter le maximum d'observations qualitatives et quantitatives sur la transformation chimique considérée.

Dans une deuxième phase, il est procédé à *l'analyse stoechiométrique, thermodynamique* et *cinétique* de la réaction. A l'issue de cette étape, il est possible de caractériser la réaction par un système d'équations stoechiométriques, les constantes d'équilibres et les chaleurs de réaction associées. On aura obtenu également des indications cinétiques précieuses, telles que la connaissance des facteurs microcinétiques et macrocinétiques qui influent sur la vitesse apparente de la transformation chimique, ainsi que des lois quantitatives reliant les vitesses aux différents facteurs cinétiques, et, en particulier, concentrations et température.

A l'issue des plans d'expérimentation et d'analyse des résultats expérimentaux, on dispose d'informations qui permettent d'aborder la *modélisation* de la réaction. De nombreux types de modèles cinétiques de la réaction chimique ont été (et sont toujours) utilisés. On peut citer les modèles purement empiriques (régression multilinéaires par exemple), les modèles à pseudo-constituants, les modèles moléculaires".

Nous nous intéresserons dans ce travail exclusivement aux modèles cinétiques fondamentaux (dits encore modèles de connaissance) de la réaction chimique que constituent les mécanismes réactionnels.

2. MECANISMES REACTIONNELS.

Nous donnons dans ce paragraphe quelques indications plus précises sur le concept de mécanisme réactionnel et sur les méthodes d'élaboration de ceux-ci. Nous illustrons notre exposé par l'exemple de la pyrolyse du néopentane, qui a largement été étudiée au laboratoire [BARONNET 1971, 1974], [RONDEAU 1976], [MARQUAIRE 1978], [AZAY 1981].

2.1. Définitions et élaboration d'un mécanisme réactionnel.

"La loi de LAVOISIER permet de définir pour toute réaction chimique un bilan matériel qui est habituellement exprimé sous la forme d'une ou de plusieurs équations. Un tel bilan ne traduit que d'une façon globale l'ensemble des phénomènes chimiques qui se produisent au niveau moléculaire et qui constituent ce que l'on appelle le mécanisme de la réaction". [METZGER 1973].

Ainsi la pyrolyse du néopentane à avancement faible est représentée essentiellement par les deux équations stoechiométriques :

$$\text{néo } C_5H_{12} = \text{iso } C_4H_8 + CH_4$$

$$2\ \text{néo } C_5H_{12} = 2\ \text{iso } C_4H_8 + C_2H_6 + H_2$$

G.M. CÔME [CÔME 1980, 1981] définit de la manière suivante les mécanismes réactionnels et propose une méthodologie d'élaboration de ceux-ci.

"Un mécanisme réactionnel est composé d'un *ensemble de processus élémentaires concomitants*. Chaque processus élémentaire décrit le déroulement

intime à l'échelle moléculaire d'un acte réactionnel irréductible. Ces processus font intervenir des intermédiaires très réactifs et très instables tels que radicaux libres, ions, molécules polaires, complexes catalytiques. Les lois de vitesse auxquelles obéissent les processus élémentaires sont fournies par la *théorie du complexe activé* ou de l'*état de transition*.

Les mécanismes réactionnels sont rarement simples, c'est-à-dire comportant un seul processus élémentaire et, éventuellement, le processus inverse. Ils correspondent le plus souvent à une succession d'étapes élémentaires, le nombre d'étapes utilisées pouvant maintenant être parfois très élevé (quelques dizaines à quelques centaines). Historiquement, l'accroissement de la complexité des mécanismes réactionnels utilisés est relativement récent (depuis 1970 environ) et a résulté essentiellement de trois causes : les progrès des méthodes expérimentales, et notamment analytiques, ont permis d'obtenir des informations de plus en plus nombreuses et précises sur les systèmes réactionnels ; les progrès de la cinétique chimique théorique ont abouti à représenter de larges classes de réactions par des mécanismes formels généraux et à proposer des critères de choix de tel ou tel processus élémentaire dans telles ou telles conditions opératoires ; c'est ainsi que l'on peut aujourd'hui décrire et classer l'ensemble des réactions de la chimie organique à partir d'une dizaine environ de processus élémentaires fondamentaux ; enfin les progrès des traitements informatiques permettent effectivement de recourir à de tels modèles complexes (cf. § 3).

L'élaboration d'un mécanisme réactionnel est une activité complexe qui utilise d'une part les résultats de l'analyse stoechiométrique, thermodynamique et cinétique (et notamment les lois de vitesse quantitatives de la réaction), ainsi que certains résultats expérimentaux particuliers (concernant par exemple la nature des intermédiaires actifs mis en jeu, la caractérisation stéréochimique des produits, etc...) et d'autre part les concepts et les modèles de la Cinétique Chimique (principe du moindre changement de structure, réaction en séquence ouverte ou fermée, étape limitant la vitesse, cinétique thermochimique, etc...)".

2.2. Exemple de la pyrolyse du néopentane à avancement quasi-nul.

Le néopentane, $CH_3-\underset{\displaystyle CH_3}{\overset{\displaystyle CH_3}{\overset{|}{\underset{|}{C}}}}-CH_3$, est un corps qui se décompose à 500°C pour donner essentiellement du méthane, CH_4, et de l'isobutène, $CH_2=C(CH_3)_2$, ainsi que des traces d'éthane, CH_3-CH_3, et d'hydrogène, H_2.

Le bilan matériel est défini par les deux équations stoechiométriques suivantes :

$$\text{néo } C_5H_{12} = \text{iso } C_4H_8 + CH_4$$

$$2\ \text{néo } C_5H_{12} = 2\ \text{iso } C_4H_8 + C_2H_6 + H_2$$

Nous donnons ici le mécanisme de cette réaction tel qu'il est admis par la plupart des auteurs.

a) La réaction s'amorce par la rupture d'une liaison C – C d'une molécule de néopentane, donnant naissance à deux radicaux libres.

$$CH_3-\underset{\displaystyle CH_3}{\overset{\displaystyle CH_3}{\overset{|}{\underset{|}{C}}}}-CH_3 \longrightarrow CH_3-\underset{\displaystyle CH_3}{\overset{\displaystyle CH_3}{\overset{|}{\underset{|}{C}}}}\cdot + \cdot CH_3$$

b) Le radical méthyle, $\cdot CH_3$, arrache un atome d'hydrogène à une molécule de néopentane.

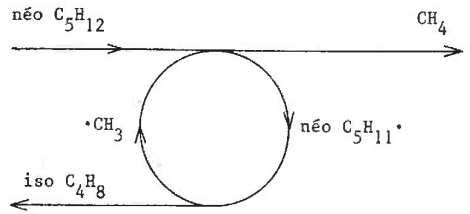
$$\cdot CH_3 + CH_3-\underset{\displaystyle CH_3}{\overset{\displaystyle CH_3}{\overset{|}{\underset{|}{C}}}}-CH_3 \longrightarrow \underline{\underline{CH_4}} + \dot{C}H_2-\underset{\displaystyle CH_3}{\overset{\displaystyle CH_3}{\overset{|}{\underset{|}{C}}}}-CH_3$$

Ce processus conduit à la formation de méthane qui est un des produits principaux de la réaction.

c) Le radical libre $\dot{C}H_2-\underset{\displaystyle CH_3}{\overset{\displaystyle CH_3}{\overset{|}{\underset{|}{C}}}}-CH_3$ (néo $C_5H_{11}\cdot$) obtenu en (b) se décompose unimoléculairement.

$$\cdot CH_2-\underset{\displaystyle CH_3}{\overset{\displaystyle CH_3}{\overset{|}{\underset{|}{C}}}}-CH_3 \longrightarrow \underline{\underline{CH_2=C(CH_3)_2}} + \dot{C}H_3$$

Ce processus donne naissance au deuxième produit principal de la réaction : l'isobutène. Il se produit également le radical $\dot{C}H_3$, qui réagit avec le néopentane dans le processus (b). Ces deux étapes consomment donc une molécule de néopentane et produisent une molécule de méthane et une molécule d'isobutène. Nous pouvons représenter ces deux étapes par le schéma suivant, caractéristique d'une réaction en chaîne.



Les deux radicaux $\cdot CH_3$ et néo $C_5H_{11}\cdot$ sont appelés porteurs de chaîne.

d) Le radical libre $CH_3 - \overset{CH_3}{\underset{CH_3}{\overset{|}{\underset{|}{C}}}}\cdot$, produit par (a) peut également se décomposer et donner de l'isobutène.

$$CH_3 - \overset{CH_3}{\underset{CH_3}{\overset{|}{\underset{|}{C}}}}\cdot \longrightarrow \cdot H + CH_2 = \underline{C\begin{matrix}CH_3\\CH_3\end{matrix}}$$

e) L'atome libre $\cdot H$ arrache un atome d'hydrogène à une molécule de néopentane pour donner de l'hydrogène.

$$\cdot H + CH_3 - \overset{CH_3}{\underset{CH_3}{\overset{|}{\underset{|}{C}}}} - CH_3 \longrightarrow \underline{H_2} + \cdot CH_2 - \overset{CH_3}{\underset{CH_3}{\overset{|}{\underset{|}{C}}}} - CH_3$$

f) Enfin l'éthane est produit par la combinaison de deux radicaux méthyle.

$$2 \cdot CH_3 \longrightarrow \underline{CH_3 - CH_3}$$

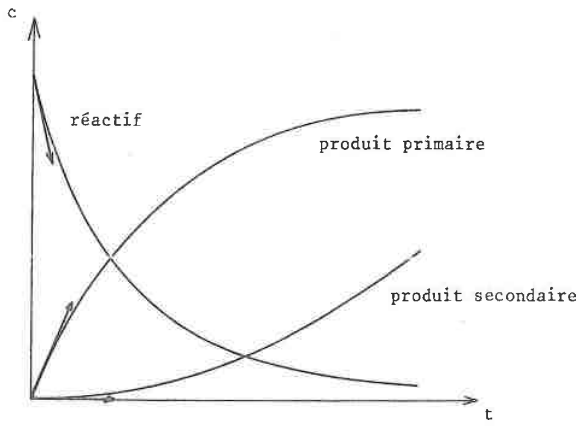
Ces six processus décrivent le mécanisme de la pyrolyse du néopentane à avancement faible. Nous pouvons, à partir de ce mécanisme, retrouver les deux équations stoechiométriques, à savoir :

$$\text{néo } C_5H_{12} = \text{iso } C_4H_8 + CH_4$$
$$2 \text{ néo } C_5H_{12} = 2 \text{ iso } C_4H_8 + C_2H_6 + H_2$$

Pour cela on fait d'une part le bilan des processus (b) et (c) et, d'autre part, le bilan des processus (a), (c), (d), (e) et (f). Le choix des processus à combiner est fait de telle sorte que le bilan élimine les radicaux libres qui sont des espèces intermédiaires peu concentrées.

Ce mécanisme n'est valable qu'à avancement quasi-nul car nous ne faisons réagir dans les processus élémentaires que le néopentane, réactif initial, et les radicaux libres, très réactifs, à l'exclusion des produits de la réaction. De ce fait ce mécanisme et les produits obtenus, isobutène, méthane, éthane et hydrogène, sont dits primaires. Dans la suite de la réaction, les produits primaires sont également susceptibles de réagir et donneront de nouveaux produits appelés secondaires.

Cette classification des produits rend compte des caractéristiques des courbes donnant la concentration c des divers constituants en fonction de la durée t de la réaction.



La pente à l'origine de la courbe c = f(t), pour un réactif, est strictement négative, car le réactif est consommé. Pour un produit primaire, la pente est strictement positive, car le produit apparaît dès le début de la réaction. Enfin pour un produit secondaire, cette pente est nulle, car un produit secondaire est formé à partir de produits primaires. Ces produits apparaissent donc quand la quantité des produits primaires est suffisante.

A partir de cet exemple, nous pouvons faire quelques remarques utiles concernant l'écriture d'un mécanisme réactionnel.

- Les radicaux libres sont des espèces intermédiaires très réactives et ont une durée de vie très courte. Dans un mécanisme il faut donc que tout radical libre produit par un processus soit consommé par un autre processus : il ne peut y avoir accumulation de radicaux libres dans le milieu réactionnel, à la différence de ce qui se passe avec des produits stables.

- Chaque processus n'apparaît qu'une seule fois dans un mécanisme. Par exemple, bien que le néopentane présente quatre liaisons C — C, nous n'écrivons qu'une seule fois le processus (a).

- Dans ce mécanisme, tous les processus a priori possibles ne sont pas décrits. Par exemple nous n'avons envisagé que la rupture d'une liaison C — C du néopentane (processus (a)) et non la rupture d'une liaison C — H. Mais des considérations thermocinétiques montrent que la rupture d'une liaison C — H est beaucoup plus difficile que la rupture d'une liaison C — C, et, par suite, le processus associé est négligeable.

2.3. Description formelle [CÔME 1980, 1981].

"Un modèle mécanistique est une séquence de processus élémentaires, chacun d'eux décrivant les actes chimiques tels qu'ils surviennent effectivement à l'échelle moléculaire. Un mécanisme réactionnel est décrit formellement comme un ensemble de s processus élémentaires irréversibles mettant en jeu les c constituants $C_1, C_2, \ldots, C_c$ présents dans le milieu réactionnel (réactifs, intermédiaires à courte durée de vie, produits, catalyseurs, inertes, ...) :

$$0 \longrightarrow \sum_{j=1}^{c} \nu_{ij} C_j \qquad ; \qquad i = 1, 2, \ldots, s$$

En accord avec la convention usuelle, les coefficients stoechiométriques ν_{ij} sont négatifs pour des réactifs et positifs pour des produits".

Exemple :

Le mécanisme de pyrolyse du néopentane à avancement nul (cf. § 2.2) est représenté par le système des six processus élémentaires suivants :

$$0 \longrightarrow - \text{néo } C_5H_{12} + \text{tert } C_4H_9^{\bullet} + CH_3^{\bullet} \qquad (a)$$

$$0 \longrightarrow - CH_3^{\bullet} - \text{néo } C_5H_{12} + CH_4 + \text{néo } C_5H_{11}^{\bullet} \qquad (b)$$

$$0 \longrightarrow - \text{néo } C_5H_{11}^{\bullet} + \text{iso } C_4H_8 + CH_3^{\bullet} \qquad (c)$$

$$0 \longrightarrow - \text{tert } C_4H_9^{\bullet} + H^{\bullet} + \text{iso } C_4H_8 \qquad (d)$$

$$0 \longrightarrow - H^{\bullet} - \text{néo } C_5H_{12} + H_2 + \text{néo } C_5H_{11}^{\bullet} \qquad (e)$$

$$0 \longrightarrow - 2\,CH_3^{\bullet} + C_2H_6 \qquad (f)$$

"Il convient de remarquer que, au moins en principe, les coefficients stoechiométriques d'un processus élémentaire sont définis de façon absolue, et non pas à un facteur de proportionnalité près, comme c'est le cas pour ceux d'une équation stoechiométrique ordinaire.

La vitesse r_i du processus n° i s'écrit :

$$r_i = k_i \prod_{j=1}^{c} c_j^{\alpha_{ij}}$$

avec $$\alpha_{ij} = \frac{|\nu_{ij}| - \nu_{ij}}{2}$$

et où
- c_j est la concentration molaire volumique du constituant C_j ;
- k_i est la constante de vitesse du processus n° i.

Habituellement, les variations de la constante de vitesse k_i en fonction de la température sont décrites par une expression d'Arrhénius :

$$k_i = A_i \exp(-E_i/RT)$$

où
- A_i est le facteur préexponentiel, positif ou nul, de la constante k_i ;
- E_i est l'énergie d'activation du processus n° i ;
- R est la constante des gaz parfaits ;
- T est la température absolue, en degré Kelvin.

Notons que la constante cinétique k, ou les paramètres d'Arrhénius A et E, ne dépendent que du processus élémentaire, c'est-à-dire qu'un même processus élémentaire, qui apparaît dans les mécanismes de deux réactions différentes, aura la même constante cinétique (à la même température) et les mêmes paramètres d'Arrhénius.

La vitesse algébrique $(R_j)_i$ de production du constituant C_j due à la ième réaction est proportionnelle à la vitesse r_i de cette $i^{ème}$ réaction, le coefficient de proportionnalité étant précisément le coefficient stoechiométrique ν_{ij} :

$$(R_j)_i = \nu_{ij}\, r_i$$

Remarquons que si ν_{ij} est négatif, c'est-à-dire si C_j est un réactif de la réaction n° i, la vitesse $(R_j)_i$ est négative, ce qui indique bien que C_j est consommé par cette réaction. Inversement si ν_{ij} est positif, la vitesse $(R_j)_i$ est positive, car C_j est produit par la réaction.

Par sommation sur les s réactions, on obtient la vitesse globale (algébrique) R_j de formation du constituant C_j :

$$R_j = \sum_{i=1}^{s} \nu_{ij}\, r_i$$

L'ensemble des processus élémentaires et des paramètres cinétiques associés (constantes de vitesse ou grandeurs d'Arrhénius) constituent un *modèle physico-chimique* de la réaction. Les expressions des vitesses R_j de formation des divers constituants en sont l'équivalent mathématique. Il est possible d'écrire le *modèle mathématique* de la réaction dès lors que l'on connaît la *matrice des coefficients stoechiométriques* ν_{ij} (i = 1 à s, j = 1 à c). Le calcul de valeurs numériques des R_j (j = 1 à c) nécessite en outre la connaissance des *paramètres cinétiques* A_i et E_i (i = 1 à s) et celle des *conditions expérimentales*, concentrations c_j (j = 1 à c) et température T.

Les conditions expérimentales peuvent être fixées par l'opérateur (c'est le plus souvent le cas de la température T) ou être calculées à partir d'autres conditions opératoires par résolution des équations de bilan de matière, d'énergie et de quantité de mouvement.

Ces relations dépendent uniquement du type du réacteur, et sont donc le résultat de la modélisation du réacteur utilisé [VILLERMAUX 1980]. Par exemple, dans le cas d'une réaction homogène qui s'effectue dans un

réacteur fermé, parfaitement agité, isotherme, isochore, le bilan de matière conduit à écrire un système d'équations différentielles ordinaires, à conditions initiales :

$$\begin{cases} \dfrac{dc_j}{dt} = R_j \,, \\ \\ c_j(0) = c_{j,o} \\ \\ j = 1, 2, \ldots, c \end{cases}$$

c_j est la concentration du constituant C_j à l'instant t, R_j la vitesse algébrique de formation de C_j déduite du mécanisme réactionnel de la manière qui vient d'être indiquée, t la durée de la réaction et $c_{j,o}$ la concentration initiale (à t = 0) du constituant C_j. $c_{j,o}$ est une variable opératoire fixée par l'expérimentateur.

Le calcul des concentrations c_j, pour des concentrations $c_{j,o}$, une température T et un temps t de réaction donnés permet de résoudre deux problèmes.

Si les paramètres cinétiques A_i et E_i (i = 1 à s) d'un mécanisme sont connus, il est possible de prédire les valeurs des c_j pour différentes valeurs des $c_{j,o}$, T et t ; on réalise ainsi une *simulation* du fonctionnement du réacteur considéré dans différentes conditions pour toute réaction de mécanisme connu.

Inversement, si certains des paramètres A_i et E_i sont inconnus, l'ajustement des valeurs théoriques des c_j prédites par le modèle et des valeurs expérimentales obtenues dans différentes conditions, permet d'estimer certains de ces coefficients. Il s'agit alors d'un problème d'*estimation paramétrique* ou encore d'*identification de modèle*.

En résumé, le modèle mathématique complet représentant le déroulement d'une réaction chimique homogène résulte de la combinaison de deux modèles construits indépendamment :

- le modèle mécanistique de la réaction, qui décrit le détail microscopique de celle-ci, et qui, de ce fait, est aussi valable au laboratoire qu'à l'échelle industrielle ; ce modèle se traduit par les équations suivantes :

$$r_i = A_i e^{-E_i/RT} \prod_{j=1}^{c} c_j^{(|\nu_{ij}| - \nu_{ij})/2} \qquad i = 1, 2, \ldots, s$$

$$R_j = \sum_{i=1}^{s} \nu_{ij} r_i \qquad j = 1, 2, \ldots, c$$

- le modèle de réacteur, qui définit les conditions de mise en oeuvre de la réaction. Par exemple, dans le cas d'un réacteur fermé, parfaitement agité, isotherme et isochore, on a les équations caractéristiques suivantes :

$$\frac{dc_j}{dt} = R_j$$

$$c_j(0) = c_{j,o}$$

$$j = 1, 2, \ldots, c$$

En conséquence, l'extrapolation d'un tel modèle, nécessaire pour passer d'une étude de laboratoire à une exploitation industrielle, n'implique que la construction d'un nouveau modèle de réacteur.

Cependant l'élaboration d'un modèle mécanistique n'est pas immédiate :

- par définition même de ce type de modèle, il faut avoir des connaissances fondamentales sur le mécanisme des réactions chimiques, et, dans ce domaine, tout n'est pas encore élucidé ;

- le niveau microscopique de la description des réactions multiplie le nombre d'équations du modèle, ce qui rend son traitement (manuel) difficile ;

- l'exploitation de ce modèle n'est possible que si l'on connaît les constantes de vitesse, sinon de tous, du moins des processus élémentaires déterminants du mécanisme, et que l'on dispose d'ordre de grandeur pour les autres.

Les récents progrès réalisés ces dernières années en cinétique chimique, notamment dans l'étude des réactions radicalaires en phase gazeuse, ont considérablement réduit, et tendent à éliminer, ces obstacles qui s'opposent à l'approche mécanistique.

En effet, les réactions en phase gazeuse peuvent être considérées comme des réactions modèles, car l'état gazeux est l'état de la matière le plus simple et le mieux connu. Des méthodes expérimentales ont été conçues pour étudier des réactions globales ou des processus élémentaires quasi-isolés dans de vastes domaines de pression et de température. Les intermédiaires à courte durée de vie ont également été mis en évidence par diverses méthodes. De très nombreux systèmes ont été étudiés, de sorte qu'il existe un large accord sur la plupart des mécanismes réactionnels et les valeurs des paramètres thermochimiques et cinétiques associés, même si quelques désaccords subsistent ; les méthodes de la cinétique thermochimique ont apporté du reste une contribution importante à l'estimation de ces grandeurs. D'autre part, sur le plan théorique, une rationalisation de l'écriture des mécanismes formels généraux a été atteinte, basée sur l'emploi de mécanismes formels généraux [GOLDFINGER 1948], [NICLAUSE 1966], [INGOLD 1953], [BURNETT 1954], et sur l'évaluation des données de cinétique thermochimique qui permettent de sélectionner les processus élémentaires susceptibles de se produire dans des conditions expérimentales données [BENSON 1976].

Ainsi, il apparaît maintenant possible d'écrire a priori des mécanismes réactionnels complexes pour des classes importantes de réactions en phase gazeuse. On trouvera dans la thèse de P. AZAY par exemple [AZAY 1981], deux cas de réactions modélisées par des mécanismes réactionnels complexes, comportant chacun des dizaines de processus élémentaires et de constituants (espèces moléculaires et radicaux libres). Il s'agit d'une part de la pyrolyse du néopentane vers 700°C et d'autre part de la chloration de l'éthylène en vue de fabriquer du chlorure de vinyle et des solvants chlorés (Procédé CHLOE)".

3. COMPILATEUR POUR LA CINETIQUE CHIMIQUE.

Il est raisonnable d'envisager le traitement d'un mécanisme complexe comportant plusieurs dizaines de processus élémentaires et d'espèces réactives avec l'aide d'ordinateurs. Ceux-ci servent, avant tout, à résoudre les problèmes d'analyse numérique et d'optimisation posés par le traitement mathématique du modèle. Mais rien que la construction et la saisie manuelle de la matrice des coefficients stoechiométriques, à partir d'un mécanisme réactionnel écrit sur le papier, est un travail fastidieux et source d'erreurs, d'autant plus qu'un mécanisme est souvent revu et corrigé au cours de son élaboration. Aussi un compilateur de langage chimique peut apporter une aide précieuse, sinon indispensable, en assurant automatiquement l'interface entre le mécanisme réactionnel et le modèle mathématique associé.

L'introduction dans l'ordinateur d'un mécanisme réactionnel et des grandeurs associées pose essentiellement un problème de notation chimique.

3.1. Notations chimiques.

Deux types de notation sont généralement employés en informatique chimique : les notations bidimensionnelles et les notations linéaires. Les notations bidimensionnelles impliquent l'utilisation de matériels sophistiqués (par exemple écran de visualisation) et de logiciels complexes. Cette notation a été utilisée dans différents travaux [WIPKE 1974], [BONNET 1979], [COREY 1972], etc...

Les notations linéaires utilisent les caractères disponibles sur le clavier des machines à écrire standards, elles n'impliquent donc pas l'utilisation de matériel spécial. La notation linéaire la plus répandue est la notation WISSWESSER [SMITH 1975]. En cinétique chimique, [EDELSON 1976] a proposé une notation linéaire très simple et a élaboré le compilateur correspondant.

S'inspirant de ce dernier travail, le laboratoire de Cinétique Appliquée, en collaboration avec le Centre de Recherche en Informatique de Nancy, a créé une notation chimique linéaire originale [ALRAN 1979].

Lors de la définition de ce langage, les auteurs avaient un objectif double. Ils voulaient d'une part un langage linéaire qui permette l'acquisition des réactions chimiques par les périphériques classiques. D'autre part, ce langage devait être aussi proche que possible des notations

habituellement utilisées par les chimistes, pour être d'un apprentissage très facile, tant en lecture qu'en écriture.

La nomenclature chimique classique utilise plusieurs types de noms pour référencer les composés chimiques [LOZAC'H 1968].

- Le nom trivial : nom dont aucune partie n'a de signification systématique. Ce nom est issu généralement des origines du composé.

Exemple : aspirine

- La formule brute : elle donne la composition atomique du composé, sans indiquer sa structure. Dans les composés simples, on peut faire précéder la formule brute d'un préfixe qui distingue les isomères.

Exemples : C_6H_6, néo C_5H_{12}

- Le nom systématique : nom formé de syllabes de signification structurale précise, qui peut comporter, ou ne pas comporter de vocables numériques.

Exemple : pentane

- Le nom semi-systématique : nom dont une partie seulement a une signification systématique.

Exemple : méthane

- La formule développée ou semi-développée : représentation bidimensionnelle de la structure, qui explicite les atomes et les liaisons du composé.

Exemple : $CH_3 - C\begin{matrix}=O\\ \diagdown OH\end{matrix}$

Le langage du compilateur doit permettre de représenter un composé sous l'une ou l'autre de ces formes, au gré de l'utilisateur.

Nous allons considérer les principaux problèmes lexicographiques qui se sont posés et la façon dont ils ont été résolus.

3.1.1. Les atomes.

Les symboles atomiques sont formés d'une majuscule, suivie éventuellement d'une minuscule.

Exemples : C, O, Br, Co, Cl

Tous les périphériques n'acceptant pas les minuscules, celles-ci sont transcrites en majuscules. Pour ne pas confondre CO, atome de cobalt, et CO, oxyde de carbone, par exemple, les symboles atomiques de plus d'une lettre sont mis entre apostrophes.

Exemples : C, O, 'BR', 'CO', 'CL'

3.1.2. Les liaisons.

Les liaisons simples, doubles et triples sont représentées respectivement par /, //, ///.

3.1.3. Les indices formulaires.

Ces indices sont utilisés dans les formules brutes ou semi-développées comme coefficient multiplicateur du symbole atomique qui précède.

Exemples : C_5H_{12}, $CH_3-CH=CH_2$

Ces indices sont simplement "remontés" au niveau de la ligne.

Exemples : C5H12, CH3/CH//CH2

3.1.4. Les charges.

Les charges des composés ioniques sont écrites en exposant.

Exemple : Cl^-, Fe^{++}, Fe^{3+}, $CH_3-CH_2^+$

Ces exposants sont "abaissés" au niveau de la ligne et mis entre parenthèses pour ne pas confondre le caractère "plus" d'une charge positive avec le caractère "plus" liant deux réactifs dans une réaction chimique, ni l'entier qui multiplie un atome avec l'entier qui multiplie une charge.

Exemples : 'CL'(-), 'FE'(++), 'FE'(3+), CH3/CH2(+)

3.1.5. Les électrons célibataires.

Les électrons célibataires, c'est-à-dire les électrons qui ne sont pas engagés dans une liaison chimique, sont représentés par des points, appelés points radicalaires, écrits indifféremment au-dessus, à gauche, ou à droite du symbole de l'atome portant ces électrons.

Exemples : $\dot{C}H_3$, $:CH_2$, $H\cdot$

Dans la notation du compilateur, ces points sont écrits à droite du symbole atomique, l'un à la suite de l'autre s'il y en a plusieurs, et mis éventuellement entre parenthèses.

Exemples : C.H3, C(.)H3 ou CH3., CH2.., H.

3.1.6. La spécification par du texte.

Le chimiste choisit souvent de décrire un corps par sa formule brute précédée ou suivie d'un préfixe ou d'un suffixe pour distinguer les isomères.

Exemple : le n-butane, $CH_3-CH_2-CH_2-CH_3$, est noté n C_4H_{10} et l'isobutane, $CH_3-CH(CH_3)-CH_3$, est noté iso C_4H_{10}

Dans la notation du compilateur, le suffixe ou le préfixe est écrit en majuscules et mis entre deux doubles apostrophes, pour les distinguer des caractères représentant les atomes.

Exemple : "N"C4H10, "ISO"C4H10

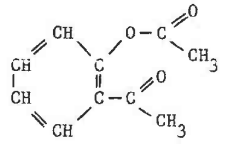
3.1.7. Groupes d'éléments.

Un constituant peut être décomposé en groupes d'éléments, séparés par des liaisons chimiques ou mis entre parenthèses. Cette possibilité est utile lorsqu'on veut transcrire linéairement la formule (semi-) développée d'un constituant.

Exemple : CH3/CH(CH3)2 est une notation possible pour l'isobutane.

La spécification par du texte, permet d'avoir une notation explicite, même pour des composés cycliques relativement complexes.

Exemple : l'aspirine dont la formule développée est :



peut être notée :

"CYCLE ORTHO" C6H4(O/C(//O)/CH3)(C(//O)/CH3)

Cette notation décrit un cycle benzénique C_6H_6 dont deux atomes d'hydrogènes sont substitués par les groupes $-O-C(=O)-CH_3$ et $-C(=O)-CH_3$ situés en Position "ortho".

3.1.8. Les processus élémentaires.

Chaque processus élémentaire est étiqueté par une suite de 1 à 5 caractères entre crochets "<" et ">". On écrit ensuite la liste des réactifs, séparés par des "+", une flèche de réaction, "->", la liste des produits, séparés par des "+" et les constantes cinétiques AD et ED.

Exemple :

<1> C(CH3)4 -> C(CH3)3. + CH3., AD=17.3, ED=82.0 ;

La souplesse de ce langage est encore accrue par la possibilité d'inclure des commentaires, écrits entre deux étoiles, et d'utiliser des déclarations qui permettent de simplifier l'écriture des mécanismes.

3.2. Résultats de la compilation.

Le compilateur de langage chimique effectue quelques vérifications sémantiques, pour chaque processus élémentaire.

- Lorsqu'il peut compter le nombre d'atomes des constituants, c'est-à-dire si la notation est dérivée de la formule brute ou (semi-) développée, il vérifie la conservation des éléments.

- Il vérifie que la somme algébrique des charges portées par les réactifs est égale à celle des charges portées par les produits.

- Il vérifie que le nombre des points radicalaires présents dans une réaction est multiple de deux (l'association de deux électrons constitue une liaison chimique).

L'échec d'un de ces tests provoque l'édition d'un message d'erreur. Cependant l'analyse du mécanisme n'est pas interrompue, car il existe des réactions qui font exception aux règles de conservation.

Aucune vérification n'est faite au niveau de la structure des constituants. L'identité de deux constituants est déterminée par l'identité de la chaîne de caractères de leur notation linéaire.

Outre ces vérifications, le compilateur fournit la liste des constituants présents dans le mécanisme en indiquant éventuellement ceux qui ont la même formule brute, la liste des processus élémentaires, la matrice des coefficients stoechiométriques.

Enfin il crée un fichier où sont stockés les résultats utilisés par le programme numérique (matrice de coefficients stoechiométriques, tableaux des paramètres cinétiques).

Le compilateur de langage chimique est une première étape dans l'utilisation de l'ordinateur pour la modélisation des réactions chimiques.

Le présent travail constitue l'étape suivante du traitement automatique des mécanismes réactionnels. Alors que dans le compilateur, les réactions chimiques sont fournies par l'utilisateur, nous voulons les faire établir automatiquement sur une liste de modèles de processus élémentaires par l'ordinateur à partir de la seule donnée des réactifs qui participent à la réaction.

CHAPITRE II

Construction automatique de mécanismes réactionnels

Le but de ce projet est la conception d'une application informatique d'aide à l'écriture de mécanismes réactionnels. La définition physico-chimique du problème a été faite en vue de modéliser les réactions radicalaires des hydrocarbures en phase gazeuse. Ce choix est justifié par G.M. CÔME de la manière suivante :

"Il existe une très grande diversité de réactions chimiques, caractérisées entre autres par leur *localisation* (réactions homogènes ou hétérogènes), leur *mode d'activation* (thermique, photochimique, catalytique, ...), la nature des *intermédiaires actifs* (radicaux libres, ions, complexes), le type de *mécanisme* (réaction en séquence ouverte, en séquence fermée, à branchement, ...), etc...

A chaque grande catégorie de réaction correspond une certaine approche de la modélisation, car il n'existe pas à l'heure actuelle de théorie unitaire permettant d'aborder avec le même point de vue fondamental les différents problèmes cinétiques posés. Il n'était donc pas raisonnable d'envisager la création d'un logiciel absolument général d'aide à la construction automatique de mécanismes réactionnels quelconques.

Pour des raisons que nous avons déjà indiquées dans le premier chapitre de ce travail (§ 2.3), le choix des *réactions radicalaires en phase gazeuse* paraissait tout-à-fait indiqué pour une première recherche dans ce domaine. Rappelons en effet que les mécanismes de ces réactions sont dans de nombreux cas convenablement élucidés et que l'on dispose en outre de banques de données de paramètres cinétiques pour un nombre élevé de processus élémentaires impliqués dans ces mécanismes. En outre, ce type de réactions présente un grand intérêt pratique, puisqu'elles interviennent dans les

pyrolyses, les chlorations, les oxydations. Enfin, l'Ecole de Cinétique de Nancy a consacré depuis longtemps une part importante de ses travaux à l'étude des réactions radicalaires en phase gazeuse".

C'est donc dans ce cadre que nous allons spécifier la nature des constituants chimiques (c'est-à-dire les objets manipulés) que nous allons traiter les modèles de processus élémentaires radicalaires en phase gazeuse (c'est-à-dire les opérateurs sur ces objets), et enfin les règles d'écriture des mécanismes réactionnels (c'est-à-dire la stratégie d'enchaînement des processus élémentaires).

1. LES CONSTITUANTS CHIMIQUES.

On peut admettre, en première approximation, qu'il existe deux grands types de liaisons chimiques : les liaisons ioniques et les liaisons covalentes. Dans ce qui suit, nous nous intéresserons essentiellement aux corps covalents, qui sont les plus nombreux en chimie organique, et par ailleurs pratiquement les seuls à intervenir en phase gazeuse.

Les liaisons de covalence sont formées par la mise en commun d'une ou de plusieurs paires d'électrons, provenant de deux atomes identiques ou différents. Ces liaisons peuvent être simples, doubles ou triples selon qu'une, deux ou trois paires d'électrons sont mises en commun.

Nous distinguons deux types de constituants : les molécules dont tous les électrons non appariés des atomes contribuent à des liaisons chimiques, et les mono-radicaux libres dont un atome porte un électron célibataire. Nous appellerons atome pointé un atome porteur d'un électron célibataire, car cet électron est toujours représenté par un point. Les mono-radicaux libres sont les seules espèces intermédiaires envisagées pour ce projet, à l'exclusion des biradicaux notamment, malgré leur rôle important en chimie-physique.

Nous appellerons valence d'un atome dans un constituant le nombre de liaisons covalentes simples que peut donner cet atome. Ainsi le carbone est tétravalent, l'hydrogène et les halogènes monovalents (CH_4, $CH_2=CH_2$, $CH\equiv CH$, CH_3Cl, etc...). Pour des raisons qui apparaîtront lors de la définition des modèles de processus élémentaires, nous imposons que toutes les occurrences d'un même symbole atomique aient toujours la même valence. Ceci revient pratiquement à ne pas considérer certaines molécules ou certains radicaux libres contenant par exemple des atomes d'oxygène.

Ainsi le processus élémentaire

$$CH_3-\dot{C}=O \longrightarrow \dot{C}H_3 + C\equiv O$$

fait intervenir un oxygène divalent (dans $CH_3\dot{C}O$) et un oxygène trivalent (dans CO).

Enfin, nous avons restreint nos investigations essentiellement aux composés acycliques, pour des raisons qui seront indiquées.

Pour décrire les modèles de processus élémentaires, nous référençons les constituants par un groupe d'un, de deux, ou de trois atomes avec leurs liaisons communes, appelé motif. A, B, C désignent des atomes quelconques,

et T un atome monovalent. Le motif d'un radical libre contient toujours l'atome pointé. On appelle liaisons en α de l'atome pointé, les liaisons de cet atome, et liaisons en β de l'atome pointé, toutes les liaisons des atomes adjacents autres que les liaisons en α.

Exemples :

A = B référence une molécule ayant une liaison double.

$\cdot A-B-C$: la liaison A—B est en α de l'atome pointé ;
α β la liaison B—C est en β de l'atome pointé.

Ce motif référence un radical libre ayant deux liaisons simples, respectivement en α et en β de l'atome pointé.

2. LES MODELES DE PROCESSUS ELEMENTAIRES.

Nous faisons une distinction entre les modèles de processus élémentaires et les processus élémentaires. La première appellation regroupe l'ensemble des règles de transformation d'un constituant par rupture et fermeture des liaisons, tandis que la deuxième appellation désigne l'application de ces règles à des constituants donnés.

Sept modèles de processus élémentaires ont été retenus pour notre projet : ce sont les processus les plus importants dans les réactions radicalaires en phase gazeuse. Pour chacun d'eux, nous donnons une description du ou des motifs impliqués, tels qu'ils ont été définis ci-dessus, un commentaire sur la rupture et la formation des liaisons mises en jeu lors de la réaction, un exemple et une notation générale des processus élémentaires associés.

2.1. Amorçage unimoléculaire.

$$A-B \longrightarrow \cdot A + \cdot B$$

Ce processus consiste en la rupture d'une liaison simple d'une molécule. Le doublet d'électrons qui forme la liaison simple se scinde en deux ; la réaction fournit deux radicaux libres.

Exemple :

$$CH_4 \longrightarrow \cdot CH_3 + \cdot H$$

En adoptant la notation r—s pour une molécule ayant une liaison simple, un processus d'amorçage unimoléculaire s'écrit comme suit :

$$\text{au} : r-s \longrightarrow .r + .s$$

2.2. Combinaison de deux radicaux libres.

$$\cdot A + \cdot B \longrightarrow A-B$$

Ce modèle de processus est exactement l'inverse de l'amorçage unimoléculaire : les deux électrons célibataires portés par les atomes A et B de deux radicaux libres s'associent pour former une liaison simple. Le résultat est une molécule.

Exemple :

$$\cdot CH_2-CH_3 + \cdot CH_3 \longrightarrow CH_3-CH_2-CH_3$$

$$\text{co} : .r + .s \longrightarrow r-s$$

2.3. Addition d'un radical libre sur une molécule insaturée.

$$A=B + \cdot C \longrightarrow \cdot A-B-C \text{ ou } \cdot B-A-C$$
$$A\equiv B + \cdot C \longrightarrow \cdot A=B-C \text{ ou } \cdot B=A-C$$

L'un des doublets d'électrons d'une liaison multiple (double ou triple) se scinde en deux : l'un des électrons s'associe avec l'électron célibataire porté par un atome C d'un radical libre pour former une liaison simple, B—C par exemple ; l'autre électron reste sur l'atome A. Les atomes A et B de la molécule jouant des rôles symétriques, il y a deux radicaux libres possibles comme résultats.

Exemple :

$$CH_2=CCl_2 + \cdot H \longrightarrow CH_3-\dot{C}Cl_2 \text{ ou } \dot{C}H_2-CHCl_2$$
$$\text{ad} : m + .r \longrightarrow .m-r$$

2.4. Décomposition d'un radical libre.

$$\cdot A-B-C \longrightarrow A=B + \cdot C$$
$$\cdot A=B-C \longrightarrow A\equiv B + \cdot C$$

C'est le modèle de processus inverse de l'addition. Un radical libre peut se décomposer s'il a une liaison simple en β de l'atome pointé et si la liaison en α correspondante est simple ou double. La liaison B—C se rompt et libère un radical libre •C ; l'autre électron va former avec l'électron célibataire de A une liaison supplémentaire entre A et B. Le résultat est une molécule insaturée et un radical libre.

Exemple :

$$\cdot CH=CH-CH_3 \longrightarrow CH\equiv C-CH_3 + \cdot H$$
$$\text{de} : .m-r \longrightarrow m + .r$$

2.5. Métathèse.

$$A-T + \cdot B \longrightarrow \cdot A + B-T$$

Le radical libre •B arrache un atome monovalent à une molécule. Le modèle de processus inverse est également une métathèse. C'est la superposition d'un amorçage unimoléculaire, produisant un radical libre •T, et de la combinaison de ce radical avec un autre radical •B.

Exemple :

$$CH_4 + \cdot Cl \longrightarrow \cdot CH_3 + HCl$$
$$\text{me} : r-T + .s \longrightarrow .r + s-T$$

2.6. Amorçage bimoléculaire.

$$A=B + C-T \longrightarrow \cdot A-B-T + \cdot C \text{ ou } \cdot B-A-T + \cdot C$$
$$A\equiv B + C-T \longrightarrow \cdot A=B-T + \cdot C \text{ ou } \cdot B=A-T + \cdot C$$

Une molécule insaturée A=B (ou A≡B) arrache un atome monovalent d'une molécule C—T. Ce modèle de processus correspond à la superposition d'un amorçage unimoléculaire, produisant un atome libre •T, et de l'addition de cet atome sur une molécule insaturée.

Les deux atomes A et B jouant des rôles symétriques, il y a deux résultats possibles. D'autre part C peut-être aussi un atome monovalent, ce qui porte à quatre le nombre de résultats possibles.

Exemple :

$$CH_2=CHCl + HCl \longrightarrow \cdot CH_2-CHCl_2 + \cdot H$$
$$\text{ou } \cdot CH_2-CH_2Cl + \cdot Cl$$
$$\text{ou } \cdot CHCl-CH_2Cl + \cdot H$$
$$\text{ou } \cdot CHCl-CH_3 + \cdot Cl$$
$$\text{ab} : m + r-T \longrightarrow .m-T + .r$$

2.7. Dismutation de deux radicaux libres.

$$\cdot A-B-T + \cdot C \longrightarrow A=B + C-T$$
$$\cdot A=B-T + \cdot C \longrightarrow A\equiv B + C-T$$

Un radical libre •C arrache un atome monovalent T d'un deuxième radical libre •A—B—T. Ce modèle de processus est la superposition d'une décomposition de •A—B—T (ou •A=B—T) libérant l'atome libre •T et de la combinaison de •T avec un autre radical. C'est l'inverse de l'amorçage bimoléculaire.

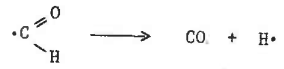
Exemple :

$$\cdot CH_2-CH_3 + \cdot CH_3 \longrightarrow CH_2=CH_2 + CH_4$$
$$\text{di} : .m-T + .r \longrightarrow m + r-T$$

Remarques

1) Ces modèles de processus élémentaires ne modifient pas la valence des atomes, puisque la suppression d'une liaison d'un atome entraîne l'apparition d'un électron célibataire sur cet atome, et réciproquement. Nous pouvons ainsi justifier le fait d'avoir imposé une valence fixe à toutes les occurrences d'un même symbole atomique.

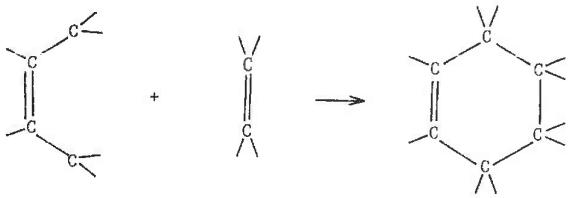
2) Le choix de ces modèles de processus limite la classe des réactions que le programme pourra traiter. Par exemple, la décomposition du radical carboxyle par le processus :



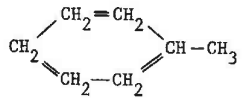
est courante dans les réactions d'oxydation, mais ce processus ne correspond à aucun des modèles retenus.

En outre, ce processus ne répond pas à la règle de valence fixe, car dans le radical  , le carbone a une valence 4 et l'oxygène une valence 2, alors que dans l'oxyde de carbone, CO, le carbone et l'oxygène devraient avoir des valences égales, d'après la définition qu'on en a donnée. Cela tient à la structure particulière de l'atome d'oxygène. En conséquence, ce genre de réactions ne pourra pas être traité par notre programme.

3) Les réactions des composés cycliques font intervenir d'autres modèles de processus élémentaires que ceux mentionnés ci-dessus. Par exemple, la cycloaddition de DIELS-ALDER est un processus moléculaire d'addition concerté de deux molécules insaturées qui construit un cycle à six carbones et que l'on peut décrire par le schéma suivant :



Mais ce type de processus est peu courant, la généralisation d'un tel processus en modèle difficile et son application délicate. Aussi, dans le cadre d'un premier prototype, avons-nous supprimé les réactions des cycles, et considéré uniquement des composés acycliques. Cependant si les cycles d'un composé ne sont pas affectés par la réaction, comme dans la chloration du toluène, on peut envisager d'en construire le mécanisme. Les cycles sont alors remplacés par un "super atome". Par exemple le toluène de formule développée :



peut être représenté par :

Ces remarques montrent qu'il n'est pas réaliste de vouloir concevoir une application informatique très générale qui puisse modéliser toutes les réactions, même en se bornant aux réactions radicalaires en phase gazeuse. Pourtant cette liste de modèles de processus élémentaires semble suffire pour écrire les mécanismes réactionnels de nombreuses réactions. Mais on ne peut caractériser ces réactions autrement que par les processus élémentaires qu'elles mettent en jeu. Il faut donc que l'utilisateur soit bien conscient des limites explicites de la méthode présentée dans ce travail.

3. STRATEGIE D'ENCHAINEMENT DES PROCESSUS ELEMENTAIRES.

Nous venons de décrire les modèles de processus élémentaires, et de caractériser les constituants auxquels seront appliqués ces modèles. La réaction à modéliser est en outre définie par une liste de molécules, appelées réactifs, fournie par l'utilisateur. Pour spécifier complètement notre application, il reste à définir une stratégie pour enchaîner les processus élémentaires.

La définition de cette stratégie résulte de considérations de divers ordres, théoriques et expérimentales. Nous nous limiterons ici à quelques indications simples extraites de [CÔME 1980, 1981] renvoyant le lecteur pour plus de détails par exemple à ces textes.

"L'analyse cinétique de nombreuses réactions radicalaires en phase gazeuse a permis assez rapidement de montrer qu'elles pouvaient fréquemment être représentées par *des mécanismes formels* assez généraux, par exemple les mécanismes βμ [GOLDFINGER 1948] et μH, YH [NICLAUSE 1966]. Par ailleurs, lors de l'élucidation du mécanisme d'une réaction donnée, les méthodes de la *cinétique thermochimique*, ainsi que les *tables de paramètres cinétiques*, permettent d'effectuer des choix parmi les différents processus élémentaires susceptibles de se produire. Enfin, les *résultats expérimentaux* eux-mêmes concernant la réaction en cours d'étude, apportant des informations décisives sur son mécanisme".

Une telle méthodologie a été utilisée récemment au Laboratoire de Cinétique Appliquée pour construire le mécanisme réactionnel complexe rendant compte de la pyrolyse du néopentane vers 700°C [AZAY 1981].

Dans ce qui suit, nous exposons tout d'abord le principe général de construction d'un mécanisme réactionnel, puis nous en donnons un énoncé formel afin de définir le problème informatique correspondant.

3.1. Principe de construction d'un mécanisme réactionnel.

On applique, systématiquement, l'ensemble des modèles de processus aux réactifs, et à tous les radicaux libres engendrés par les processus. Les molécules engendrées, qui sont les produits primaires de la réaction, sont supposées inertes dans cette première étape. Le mécanisme ainsi obtenu est appelé mécanisme primaire.

Pour obtenir un mécanisme secondaire, il faut faire réagir les réactifs initiaux, les produits primaires, et l'ensemble de tous les radicaux libres.

On obtiendrait de même les mécanismes tertiaires, quaternaires, etc..., en faisant réagir, en plus des réactifs et des produits primaires, les produits secondaires, tertiaires, etc...

Cependant, il y a lieu de craindre une inflation galopante du nombre de processus élémentaires avec le rang du mécanisme (primaire, secondaire, etc...). Il apparaît donc raisonnable de procéder d'une manière progressive :

- écriture du mécanisme primaire, confrontation avec les résultats expérimentaux, et éventuellement simplification de ce mécanisme ;
- en cas d'échec, on passe au mécanisme secondaire ;
- on répète cette dernière étape en construisant successivement le mécanisme tertiaire, quaternaire, etc... jusqu'à obtenir un mécanisme satisfaisant.

Dans la plupart des cas, le mécanisme secondaire ou tertiaire s'avère suffisant.

Dans notre travail, nous allons suivre cette démarche : nous construirons d'abord le mécanisme primaire, puis les mécanismes de rangs supérieurs à la demande de l'utilisateur. Suite à une remarque faite au paragraphe 2.2. du chapitre I, nous porterons notre attention sur le fait que, dans un mécanisme réactionnel, un même processus élémentaire ne doit évidemment apparaître qu'une seule fois.

3.2. Enoncé formel de la construction d'un mécanisme réactionnel.

Pour l'instant, le problème à résoudre est posé en termes d'objets et d'opérateurs sur ces objets : les objets sont les constituants chimiques parmi lesquels on distingue les molécules et les radicaux libres ; les opérateurs sont les sept modèles de processus élémentaires. Nous adoptons la notation globale π pour tous ces opérateurs.

Pour aboutir à un énoncé formel de la construction d'un mécanisme réactionnel, nous introduisons la notion de génération.

Définition 1.

Soit M_o un ensemble de molécules ; la génération d'origine M_o est obtenue en écrivant tous les processus possibles qui font réagir les molécules de M_o et les radicaux libres produits à partir de ces molécules ; toutes les nouvelles molécules produites sont considérées inertes.

Le résultat d'une génération est un triplet $g(M_o) = (\pi(M_o), \mu(M_o), \rho(M_o))$ où $\pi(M_o)$ est l'ensemble des processus élémentaires obtenus et $\mu(M_o)$ (respectivement $\rho(M_o)$) est l'ensemble de toutes les molécules (respectivement de tous les radicaux libres) présents dans les processus élémentaires de $\pi(M_o)$.

Remarques :

1) Nous confondrons une génération d'origine M_o avec son résultat.

2) L'ensemble $\mu(M_o)$ contient l'ensemble M_o.

$\pi(M_o)$ est, par définition, le mécanisme primaire de la réaction des molécules de M_o. $\mu(M_o) - M_o$ est l'ensemble des produits primaires. Nous appellerons respectivement résultat moléculaire et résultat radicalaire de la génération $g(M_o)$ les ensembles $\mu(M_o)$ et $\rho(M_o)$.

Pour obtenir un mécanisme secondaire complet, il faut faire réagir toutes les molécules du mécanisme primaire, c'est-à-dire tous les éléments de $\mu(M_o)$. Cela correspond à construire la génération $g(\mu(M_o))$. Par convention nous notons $g^2(M_o) = (\pi^2(M_o), \mu^2(M_o), \rho^2(M_o))$ cette génération. De proche en proche, nous pouvons définir une génération $g^n(M_o)$, appelée génération n d'origine M_o.

Définition 2.

Soient M_o un ensemble de molécules, et n un entier positif. La génération n d'origine M_o est

- la génération $g(M_o)$, si $n = 1$;

- la génération $g(\mu^{n-1}(M_o))$, où $\mu^{n-1}(M_o)$ est le résultat moléculaire de la génération $g^{n-1}(M_o)$, si $n > 1$.

Le résultat de la génération n d'origine M_o est noté : $g^n(M_o) = (\pi^n(M_o), \mu^n(M_o), \rho^n(M_o))$.

$\pi^n(M_o)$ définit le mécanisme de rang n et $\mu^n(M_o) - \mu^{n-1}(M_o)$ est l'ensemble des produits de rang n, pour la réaction des éléments de M_o.

Par convention, $\mu^o(M_o) = M_o$.

Exemple 1 :

Cherchons la génération $g(M_o) = g(\{CH_3-CH_3\})$. Avec CH_3-CH_3, nous pouvons écrire deux amorçages unimoléculaires

(1) $CH_3-CH_3 \longrightarrow \dot{H} + \dot{C}H_2-CH_3$
(2) $CH_3-CH_3 \longrightarrow 2\ \dot{C}H_3$

Seul le radical $\dot{C}H_2-CH_3$ peut se décomposer

(3) $\dot{C}H_2-CH_3 \longrightarrow CH_2=CH_2 + H\cdot$

Ce processus produit la molécule $CH_2=CH_2$, mais nous la considérons inerte.

Les métathèses que nous pouvons écrire sont :

(4) $CH_3-CH_3 + \dot{H} \longrightarrow \dot{C}H_2-CH_3 + H_2$
(5) $CH_3-CH_3 + \dot{C}H_3 \longrightarrow \dot{C}H_2-CH_3 + CH_4$
(6) $CH_3-CH_3 + \dot{C}H_2-CH_3 \longrightarrow \dot{C}H_2-CH_3 + CH_3-CH_3$

Avec les radicaux $\dot{H}$, $\dot{C}H_3$, $\dot{C}H_2-CH_3$, nous pouvons écrire les combinaisons :

(7) $2\ \dot{H} \longrightarrow H_2$
(8) $\dot{H} + \dot{C}H_3 \longrightarrow CH_4$
(9) $\dot{H} + \dot{C}H_2-CH_3 \longrightarrow CH_3-CH_3$
(10) $2\ \dot{C}H_3 \longrightarrow CH_3-CH_3$
(11) $\dot{C}H_3 + \dot{C}H_2-CH_3 \longrightarrow CH_3-CH_2-CH_3$
(12) $2\ \dot{C}H_2-CH_3 \longrightarrow CH_3-CH_2-CH_2-CH_3$

et les dismutations :

(13) $\dot{C}H_2-CH_3 + \dot{H} \longrightarrow CH_2=CH_2 + H_2$
(14) $CH_3-\dot{C}H_2 + \dot{C}H_3 \longrightarrow CH_2=CH_2 + CH_4$
(15) $2\ CH_3-\dot{C}H_2 \longrightarrow CH_2=CH_2 + CH_3-CH_3$

Les résultats moléculaires et radicalaires de cette génération sont les suivants :

$\mu(M_o) = \{CH_3-CH_3, CH_2=CH_2, H_2, CH_4, CH_3-CH_2-CH_2-CH_3, CH_3-CH_2-CH_3\}$

$\rho(M_o) = \{CH_3-\dot{C}H_2, H\cdot, \dot{C}H_3\}$

La génération $g^2(\{CH_3-CH_3\})$ est obtenue en construisant $g(\{CH_3-CH_3, CH_2=CH_2, H_2, CH_4, CH_3-CH_2-CH_2-CH_3, CH_3-CH_2-CH_3\})$.

Remarque :

Si aucune des molécules de M_o n'a de liaison simple, alors aucun processus d'amorçage n'est possible, et on ne peut pas produire de radicaux libres. Dans ce cas nous avons :

$$\pi(M_o) = \emptyset \; ; \; \mu(M_o) = M_o \; ; \; \rho(M_o) = \emptyset$$

Par suite, les générations $g^n(M_o)$, pour tout entier n, sont identiques.

Exemple 2 :

Cherchons la génération $g^1(M_o) = g(\{CH_2 = CH_2\})$.

Amorçage unimoléculaire avec $CH_2 = CH_2$

(1) $CH_2 = CH_2 \longrightarrow \dot{C}H = CH_2 + H\cdot$

Amorçage bimoléculaire avec $CH_2 = CH_2$

(2) $2\ CH_2 = CH_2 \longrightarrow \dot{C}H = CH_2 + \dot{C}H_2 - CH_3$

Décompositions avec $\dot{C}H = CH_2$ et $\dot{C}H_2 - CH_3$

(3) $CH_2 = \dot{C}H \longrightarrow CH \equiv CH + \dot{H}$

(4) $CH_3 - \dot{C}H_2 \longrightarrow CH_2 = CH_2 + H\cdot$

Addition sur $CH_2 = CH_2$ de $\dot{H}$, $\dot{C}H = CH_2$, et $\dot{C}H_2 - CH_3$

(5) $CH_2 = CH_2 + \dot{H} \longrightarrow \dot{C}H_2 - CH_3$

(6) $CH_2 = CH_2 + \dot{C}H = CH_2 \longrightarrow \dot{C}H_2 - CH_2 - CH = CH_2$

(7) $CH_2 = CH_2 + \dot{C}H_2 - CH_3 \longrightarrow \dot{C}H_2 - CH_2 - CH_2 - CH_3$

etc...

Les processus d'addition (6) et (7) produisent deux nouveaux radicaux, $\dot{C}H_2 - CH_2 - CH = CH_2$ et $\dot{C}H_2 - CH_2 - CH_2 - CH_3$, qui peuvent à leur tour s'additionner sur $CH_2 = CH_2$ et produire de nouveaux radicaux.

Une telle succession des processus d'addition correspond à des réactions de polymérisation. En pratique, le nombre d'additions consécutives peut être très élevé, ce qui se traduit par un haut degré de polymérisation. Il faut donc limiter l'application de l'addition. Ceci peut se faire selon plusieurs critères :

(a) on limite la taille des radicaux, en fixant :

(a1) un nombre limite d'atomes ;

(a2) un nombre limite d'atomes multivalents, c'est-à-dire de valence supérieure à un ;

(a3) un nombre limite d'atomes de carbone ;

(b) on limite la classe des radicaux qui participent à une addition ;

(c) on limite le nombre d'additions successives sur la même molécule.

La solution (b) a été rejetée car la caractérisation des radicaux candidats aux additions est difficile à généraliser d'un point de vue physico-chimique.

La solution (c) n'est pas valable car la taille, et par suite le nombre des molécules et des radicaux croissent de façon importante avec le rang du mécanisme.

La solution (a3) n'élimine pas la possibilité (formelle) d'avoir une infinité d'additions avec les réactifs $\{N \equiv N$ et $H_2\}$.

Aucune raison physico-chimique ne semble donner la préférence à l'une des solutions (a1) ou (a2) et nous avons choisi la première. Cette seule condition sur les radicaux produits par les additions, suffit à rendre fini le nombre de processus et de constituants d'une génération (M_o étant évidemment fini).

Nous donnons les définition et proposition suivantes :

Définition 3 :

Soient M_o un ensemble de molécules et t un entier positif ; la génération limitée par t, d'origine M_o est obtenue à partir de la génération d'origine M_o, en supprimant tous les processus d'addition produisant des radicaux libres de taille supérieure à t, et donc aussi tous les processus faisant réagir ces radicaux. Nous notons $g_t(M_o) = (\pi_t(M_o), \mu_t(M_o), \rho_t(M_o))$ le résultat de cette génération.

Proposition 1 :

Si M_o est un ensemble de cardinal fini, t et n deux entiers positifs, les trois ensembles $\pi_t^n(M_o)$, $\mu_t^n(M_o)$ et $\rho_t^n(M_o)$ de la génération n limitée par t d'origine M_o sont de cardinal fini.

Démonstration :

Notons a(c) la taille d'un constituant c ($a(c) \in \mathbb{N}^*$) et soit $u = \max_{c \in M_o} a(c)$; u est évidemment fini.

Montrons d'abord que la taille des radicaux libres de $\rho_t(M_o)$ est majorée.

- La taille des radicaux libres, .r et .s, produits par l'amorçage unimoléculaire (au : r—s ⟶ .r + .s) est majorée par u - 1.
- La taille des radicaux libres, .m—T et .r, produits par l'amorçage bimoléculaire (ab : m + r—T ⟶ .m—T + .r) est majorée par u + 1.
- La taille du radical, .r, produit par la métathèse (me : r—T + .s ⟶ .r + s—T) est majorée par u - 1.
- La taille maximum d'un radical libre, .m—r, produit par une addition est fixée à t.
- Enfin, la décomposition d'un radical .m—r, produit toujours un radical .r de taille strictement inférieur à .m—r.

Donc tous les radicaux libres de $\rho_t(M_o)$ ont une taille majorée par sup (u + 1,t).

En conséquence l'ensemble $\rho_t(M_o)$ est de cardinal fini. Par suite les processus de la génération $g_t(M_o)$, qui font réagir l'ensemble fini de constituants $M_o \cup \rho_t^1(M_o)$, sont en nombre fini.

La même analyse de la taille des molécules produites par les processus élémentaires de $\pi(M_o)$ montre qu'elle est majorée par 2 sup (u + 1,t). Donc l'ensemble $\mu_t(M_o)$ est de cardinal fini.

Un raisonnement par récurrence montre que les ensembles $\pi_t^n(M_o)$, $\mu_t^n(M_o)$, $\rho_t^n(M_o)$ de la génération n sont également finis. □

Dans notre application, nous allons, bien sûr, ne construire que des générations limitées. Le programme ne sera donc pas utilisable pour des réactions de polymérisation (non bornées). Le choix de la taille limite t des radicaux libres est laissé à l'utilisateur.

Dans la suite, nous ne considérons que des générations limitées et nous les appelons simplement générations.

3.3. Rang d'un processus élémentaire.

Etant donné un ensemble fini M_o de molécules et t un entier positif, nous avons défini les générations $g_t^n(M_o)$ d'origine M_o, pour n entier positif. Par construction, la suite $(\mu_t^n(M_o))_n$ des résultats moléculaires est une suite croissante, au sens de l'inclusion. Il est clair que cela entraîne que les suites $(\pi_t^n(M_o))_n$ et $(\rho_t^n(M_o))_n$ sont aussi croissantes. Cela signifie en particulier que, à la génération n, on retrouve tous les processus de la génération n - 1, qu'il est inutile de reconstruire.

Pour établir un mécanisme réactionnel, nous construirons donc successivement les ensembles de processus :

$$\pi_t(M_o),\ \pi_t^n(M_o) - \pi_t^{n-1}(M_o),\ n > 1$$

au lieu de construire les ensembles $\pi_t^n(M_o)$.

Nous appelons processus primaires les éléments de $\pi_t(M_o)$ et processus de rang n les éléments de $\pi_t^n(M_o) - \pi_t^{n-1}(M_o)$ (n > 1).

Une caractérisation simple des processus de rang n s'obtient en considérant chaque modèle de processus comme une fonction. Dans le tableau suivant nous donnons l'ensemble source pour chaque modèle : M désigne l'ensemble de toutes les molécules, R l'ensemble de tous les radicaux libres.

modèle de processus	ensemble source
au : amorçage unimoléculaire	M
ab : amorçage bimoléculaire	$M \times M$
de : décomposition	R
ad : addition	$M \times R$
me : métathèse	$M \times R$
co : combinaison	$R \times R$
di : dismutation	$R \times R$

Tableau 1

Si mp désigne un modèle de processus élémentaire (mp ∈ {au,ab,de,ad,me,co,di}), tous les processus de modèle mp de la génération n d'origine M_o sont obtenus en appliquant mp à l'ensemble source, noté $S(mp_t^n(M_o))$, construit à partir de $\mu_t^{n-1}(M_o)$ et $\rho_t^n(M_o)$. Par exemple pour l'addition nous avons :

$$S(ad_t^n(M_o)) = \mu_t^{n-1}(M_o) \times \rho_t^n(M_o)$$

Les processus de rang n ($n \geqslant 2$) sont, par définition, ceux de la génération n qui n'appartiennent pas à la génération n - 1. En conséquence, on les obtient en appliquant chaque modèle mp à l'ensemble de $S(mp_t^n(M_o)) - S(mp_t^{n-1}(M_o))$. Par exemple pour l'addition cet ensemble est :

$$S(ad_t^n(M_o)) - S(ad_t^{n-1}(M_o)) = [\mu_t^{n-1}(M_o) \times \rho_t^n(M_o)] - [\mu_t^{n-2}(M_o) \times \rho_t^{n-1}(M_o)]$$

Pour avoir une expression des ensembles sources plus adaptée à un énoncé algorithmique, nous posons :

$$M_n = \mu_t^n(M_o) - \mu_t^{n-1}(M_o) \qquad n \geqslant 1$$

$$R_1 = \rho_t(M_o)$$

$$R_n = \rho_t^n(M_o) - \rho_t^{n-1}(M_o) \qquad n \geqslant 2$$

M_n, $n > 0$, est l'ensemble des molécules de rang n (les molécules de rang 0 sont les réactifs initiaux), et par analogie nous dirons que les éléments de R_n, $n > 1$, sont les radicaux de rang n.

Nous obtenons alors pour l'addition :

$$S(ad_t^n(M_o)) - S(ad_t^{n-1}(M_o)) = [M_{n-1} \times \rho_t^{n-1}(M_o)] + [\mu_t^{n-1}(M_o) \times R_n],$$

ce qui exprime que, à la génération n, il faut considérer

- l'addition des radicaux de rang $i \leqslant n-1$ ($\rho_t^{n-1}(M_o)$) sur les molécules de rang n - 1 (M_{n-1})
- l'addition des radicaux de rang n (R_n) sur les molécules de rang $i \leqslant n-1$ ($\mu_t^{n-1}(M_o)$).

Remarque :

L'opérateur "+" est utilisé au lieu de "∪" entre les deux ensembles $M_{n-1} \times \rho_t^{n-1}(M_o)$ et $\mu_t^{n-1}(M_o) \times R_n$, car ceux-ci sont disjoints. Ceci nous assure que les processus d'addition du premier lot sont distincts des processus du second lot.

Le tableau ci-dessous donne l'expression des ensembles sources pour chaque modèle de processus, au rang 1 et aux rangs n, $n \geqslant 2$.

modèle de processus	rang 1	rang n ($n \geqslant 2$)
au	M_o	M_{n-1}
ab	$M_o \times M_o$	$[M_{n-1} \times \mu_t^{n-2}(M_o)] + [\mu_t^{n-1}(M_o) \times M_{n-1}]$
de	R_1	R_n
ad	$M_o \times R_1$	$[M_{n-1} \times \rho_t^{n-1}(M_o)] + [\mu_t^{n-1}(M_o) \times R_n]$
me	$M_o \times R_1$	$[M_{n-1} \times \rho_t^{n-1}(M_o)] + [\mu_t^{n-1}(M_o) \times R_n]$
co	$R_1 \times R_1$	$[R_n \times \rho_t^{n-1}(M_o)] + [R_n \times R_n]$*
di	$R_1 \times R_1$	$[R_n \times \rho_t^{n-1}(M_o)] + [\rho_t^n(M_o) \times R_n]$

Tableau 2

* Dans un processus de combinaison, les deux radicaux jouent des rôles symétriques. Donc les deux processus :

$$co : .r + .s \longrightarrow r-s$$

$$co : .s + .r \longrightarrow r-s$$

sont les mêmes.

C'est pourquoi l'ensemble source de la combinaison est

$$[R_n \times \rho_t^{n-1}(M_o)] + [R_n \times R_n]$$

au lieu de :

$$[R_n \times \rho_t^{n-1}(M_o)] + [\rho_t^n(M_o) \times R_n]$$

Pour la même raison, il ne faut considérer qu'un des couples $(.r,.s)$ et $(.s,.r)$ du produit cartésien $R_n \times R_n$, $n \geqslant 1$.

Il n'en est pas de même pour les amorçages bimoléculaires et les dismutations car les deux molécules (respectivement les deux radicaux) qui réagissent jouent des rôles distincts. Par exemple, les deux processus :

ab : $CH_2=CH_2 + CCl_2=CCl_2 \longrightarrow \cdot CH_2-CH_2Cl + \cdot CCl=CCl_2$

ab : $CH_2=CH_2 + CCl_2=CCl_2 \longrightarrow \cdot CH=CH_2 + \cdot CHCl_2-CCl_2$

sont différents.

Nous pouvons, dès maintenant donner une première version de l'algorithme du programme appelé *mécanisme*.

- Les molécules de M_o, et la taille maximum des radicaux libres produits par les additions, t, sont initialisées par lecture.
- On construit ensuite le mécanisme primaire, c'est-à-dire les ensembles M_1, R_1, P_1. Les réactifs des processus de P_1 sont les éléments de M_o et R_1.
- Puis, à la demande d'un utilisateur, on complète progressivement le mécanisme avec les processus de rang n. Cela revient à construire les ensembles M_n, R_n et P_n :

  - les amorçages unimoléculaires font réagir les molécules de M_{n-1} ;

  - les amorçages bimoléculaires font réagir d'abord les molécules de M_{n-1} avec celles de $\mu_t^{n-2}(M_o)$, puis les molécules de $\mu_t^{n-1}(M_o)$ avec celles de M_{n-1} ;

  - les décompositions font réagir les radicaux de R_n ;

  - les additions et les métathèses font réagir les molécules de M_{n-1} avec les radicaux de $\rho_t^{n-1}(M_o)$, puis les molécules de $\mu_t^{n-1}(M_o)$ avec les radicaux de R_n ;

  - les combinaisons font réagir les radicaux de R_n avec ceux de $\rho_t^{n-1}(M_o)$, puis les radicaux de R_n entre eux ;

  - les dismutations font réagir les radicaux de R_n avec ceux de $\rho_t^{n-1}(M_o)$, puis les radicaux de $\rho_t^{n-1}(M_o)$ avec ceux de R_n.

Les processus, les molécules et les radicaux sont stockés dans des tableaux *ensproc*, *ensmol* et *ensrad* au fur et à mesure de leur création. Tous les éléments de ces tableaux sont distincts. La dimension de *ensproc* est fixée par la constante *maxpr*, celles de *ensmol* et de *ensrad* par *maxenst*. Les variables *lastp*, *lastm* et *lastr* repèrent les indices des derniers éléments de chaque tableau. Elles sont gérées par les procédures de stockage des processus, molécules et radicaux. En outre les indices de la dernière molécule et du dernier radical de rang n sont conservés respectivement dans les tableaux : *lastmol*[*-1..maxméca*] et *lastrad*[*0..maxméca*] où *maxméca* est une constante fixant le rang maximum d'un mécanisme ; *lastmol*[*-1*] et *lastrad*[*0*] contiennent zéro. Ainsi, l'ensemble M_n, des molécules de rang n est défini par :

$$M_n = \{ensmol[i],\ i \in [lastmol[n-1]+1,\ lastmol[n]]\}$$

Le premier indice de *lastmol* est -1, car les réactifs sont, par convention, les molécules de rang 0. *lastmol*[*0*] et *ensmol*[*i*], *i* de *lastmol*[*-1*] à *lastmol*[*0*], sont initialisés par la lecture des réactifs.

Dans le programme *mécanisme*, nous aurons besoin des types suivants :

```
type indxenst : 0..maxenst ;
     indxpr : 0..maxpr ;
     indxméca : 0..maxméca ;
     % molécule : à définir ; %
     % radical  : à définir ; %
     % processus : à définir ; %
```

Les processus primaires sont construits par une procédure *mécaprim* dont l'interface est :

```
procédure mécaprim (in t : entier ;
                    in minsup : indxenst ;
                    inout lastm, lastr : indxenst ;
                    inout lastp : indxpr) ;
```

- *t* est la taille maximum des radicaux produits par les processus d'addition ;
- l'intervalle [*1, minsup*] définit M_o ;
- *lastm*, *lastr* et *lastp* sont les indices des derniers éléments respectivement de *ensmol*, *ensrad*, *ensproc* ; donc après l'appel de *mécaprim* l'intervalle [*minsup+1, lastm*] définit M_1, [*1, lastr*] définit R_1, et [*1, lastp*] définit P_1.

Les processus de rang n, $n \geq 2$, sont construits par une procédure *mécagen*, dont l'interface est :

```
procédure mécagen (in t : entier ;
                   in mininf, minsup, rinsup : indxenst ;
                   inout lastm, lastr : indxenst ;
                   inout lastp : indxpr) ;
```

- *t* a la même signification que dans mécaprim ;
- [*mininf, minsup*] définit M_{n-1} et donc [*1, mininf-1*] définit $\mu_t^{n-1}(M_o)$;
- [*1, rinsup*] définit $\rho_t^{n-1}(M_o)$;
- *lastm*, *lastr* et *lastp* ont la même signification que dans *mécaprim* : après l'appel de *mécagen*, au rang n, [*minsup+1, lastm*] définit M_n et [*rinsup+1, lastr*] définit R_n.

Avec toutes ces données l'algorithme du programme *mécanisme* est le suivant :

```
programme mécanisme ;
const maxcnst, maxpr, maxméca ;
var t : entier ; n : indxméca ;
    réponse : car ;
    ensproc : tableau [1..maxpr] de processus ;
    ensmol  : tableau [1..maxcnst] de molécule ;
    ensrad  : tableau [1..maxcnst] de radical ;
    lastm, lastr : indxcnst ;
    lastp : indxpr ;
    lastmol : tableau [-1..maxméca] de indxcnst ;
    lastrad : tableau [indxméca] de indxcnst ;

début
  lastm := 0 ; lastr := 0 ; lastp := 0 ; % initialisations %
  lastmol[-1] := 0 ; lastrad[0] := 0 ;
  n := 1 ;
  liréactifs (lastm) ; lastmol[0] := lastm ;
  imprimer (1, lastm) ;          % molécules réactifs %
  lire (t) ;                     % mécanisme primaire %
  mécaprim (t, lastmol[0], lastm, lastr, lastp) ;
  lastmol[1] := lastm ; lastrad[1] := lastr ;
  écrire ('voulez-vous le mécanisme suivant ? (o/n)') ;
  lire (réponse) ;
  tantque réponse = 'o' faire
    n := n+1 ;
    mécagen (t, lastmol[n-2]+1, lastmol[n-1], lastrad[n-1],
             lastm, lastr, lastp) ;
    écrire ('voulez-vous le mécanisme suivant ? (o/n)') ;
    lire (réponse) ;
  ftantque ;
fin. % mécanisme %
```

Il convient de remarquer qu'il est inutile de connaître l'ensemble des processus de rang strictement inférieur à n, pour créer les processus de rang n, car leur rang est lié au rang des constituants.

3.4. Ordre de prise en compte des modèles de processus.

Les procédures *mécaprim* et *mécagen* doivent chercher tous les processus du mécanisme de rang n. Nous allons créer ces processus modèle par modèle, en ayant choisi un ordre sur ces modèles. Cet ordre est défini à partir des deux constatations suivantes :

- les processus de modèles ab, me, di peuvent être créés à partir de deux processus de modèles au, ad, de, co (nous appellerons processus simples les processus de modèles au, ad, de et co et processus composés les processus de modèles ab, me, di) ;

- dans le mécanisme de rang n, l'ensemble des réactifs moléculaires est fixe, mais l'ensemble des réactifs radicalaires évolue suite à la création des processus de rang n.

3.4.1. Construction des processus composés à partir des processus simples.

Nous allons montrer qu'un processus composé de rang n peut être obtenu en superposant deux processus simples de rang inférieur ou égal à n, dont l'un au moins est de rang n. Cela entraîne que les processus composés ne produisent pas de nouveaux constituants par rapport aux processus simples. Mais cette superposition n'est que formelle, et les processus composés ont une influence propre pour la cinétique de la réaction. Ils doivent donc figurer dans un mécanisme réactionnel.

Nous aurons besoin des relations suivantes qui lient le rang d'un processus à ceux des constituants qui réagissent. Ces relations sont des conséquences immédiates de l'expression des ensembles sources des modèles de processus au rang n (tableau 2 § 3.3.).

rang(au : r−s ⟶ .r + .s) = rang(r−s) + 1
rang(ab : m + r−T ⟶ .m−T + .r) = sup {rang(m), rang(r−T)} + 1
rang(de : .m−r ⟶ m + .r) = rang(.m−r)
rang(ad : m + .r ⟶ .m−r) = sup {rang(m) + 1, rang(.r)}
rang(me : r−T + .s ⟶ .r + .s−T) = sup {rang(r−T) + 1, rang(.s)}
rang(co : .r + .s ⟶ r−s) = sup {rang(.r), rang(.s)}
rang(di : .m−T + .r ⟶ m + r−T) = sup {rang(.m−T), rang(.r)}

- Amorçage bimoléculaire.

La forme générale d'un processus d'amorçage bimoléculaire est :

$$ab : m + r-T \longrightarrow .m-T + .r$$

Ce processus est la superposition des deux processus suivants :

$$au : \quad r-T \longrightarrow .r + .T$$
$$ad : \quad m + .T \longrightarrow .m-T$$

Cherchons la relation entre le rang du processus de modèle ab et le rang des processus de modèles au et ad.

$$rang(ab : m + r-T \longrightarrow .m-T + .r) = \sup \{rang(m), rang(r-T)\} + 1$$
$$rang(au : r-T \longrightarrow .r + .T) = rang(r-T) + 1$$
$$rang(ad : m + .T \longrightarrow .m-T) = \sup \{(rang(m) + 1, rang(.T)\}$$

Or .T est un atome libre. Il existe donc dans M_o une molécule de la forme s-T, ce qui implique que le processus

$$au : \quad s-T \longrightarrow .s + .T$$

est de rang 1. Par suite :

$$rang(.T) = 1$$

Donc :

$$rang(ad : m + .T \longrightarrow .m-T) = rang(m) + 1$$

et nous avons :

$$rang(ab : m + r-T \longrightarrow .m-T + .r) =$$
$$\sup \{rang(au : r-T \longrightarrow .r + .T), rang(ad : m + .T \longrightarrow .m-T)\}$$

- Métathèse.

Le processus de métathèse :

$$me : r-T + .s \longrightarrow .r + s-T$$

est la superposition du processus d'amorçage unimoléculaire :

$$au : \quad r-T \longrightarrow .r + .T$$

et du processus de combinaison :

$$co : \quad .s + .T \longrightarrow s-T$$

Le même raisonnement que précédemment montre que

$$rang(me : r-T + .s \longrightarrow .r + s-T) =$$
$$\sup \{rang(au : r-T \longrightarrow .r + .T), rang(co : .s + .T \longrightarrow s-T)\}$$

- Dismutation :

Le processus de dismutation :

$$di : .m-T + .s \longrightarrow m + s-T$$

est la superposition du processus de décomposition :

$$de : \quad .m-T \longrightarrow m + .T$$

et du processus de combinaison :

$$co : \quad .s + .T \longrightarrow s-T ,$$

et nous avons, de même que précédemment :

$$rang(di : .m-T + .s \longrightarrow m + s-T) =$$
$$\sup \{rang(de : .m-T \longrightarrow m + .T), rang(co : .s + .T \longrightarrow s-T)\}$$

En conséquence de ces résultats, les processus composés seront construits à partir des processus simples. D'autre part, comme ils ne produisent pas de nouveaux constituants, ils n'influent pas sur l'ordre de prise en compte des processus simples.

3.4.2. Ordre de prise en compte des processus simples.

Comme nous l'avons déjà noté, l'ensemble des réactifs moléculaires au rang n est fixe, mais une partie des réactifs radicalaires est produite par les processus de rang n. En conséquence nous créerons d'abord les amorçages unimoléculaires qui produisent des radicaux à partir de molécules, et nous terminerons par les combinaisons qui ne produisent que des molécules. Quant aux processus d'addition et de décomposition, ils produisent tous deux des radicaux à partir de réactifs radicalaires. En fait, nous allons montrer que tous les radicaux produits par les décompositions de rang n ont déjà été produits par les amorçages et les additions de rang inférieur ou égal à n. Donc en ayant traité d'abord les amorçages unimoléculaires de rang n, ensuite les additions de rang n, nous sommes assurés d'aborder les décompositions de rang n puis les combinaisons de rang n avec tous les réactifs radicalaires de rang n.

Proposition 2 :

Tout radical produit par une décomposition de rang n est un radical de rang n - 1, ou un radical produit par un amorçage unimoléculaire ou une addition de rang n.

Démonstration :

Soient :

- $(\mu_t^{n-1}(M_o), \rho_t^{n-1}(M_o))$ le résultat de la génération n - 1 ;
- R_{au} l'ensemble des radicaux produits par un amorçage de rang n :

$$R_{au} = \{.u \mid \exists\ au : u-v \longrightarrow .u + .v,\ u-v \in M_{n-1}\} ;$$

- R_{ad} l'ensemble des radicaux produits par une addition de rang n de la forme :

ad : $m' + .u \longrightarrow .m'-u$

avec : $.u \in \rho_t^{n-1}(M_o) \cup R_{au} \cup R_{ad}$

et $m' \in \mu_t^{n-1}(M_o)$;

$R_{ad} = \{.m'-u \mid \exists$ ad : $m' + .u \longrightarrow .m'-u,\ a(.m'-u) < t$

et $(m', .u) \in [M_{n-1} \times \rho_t^{n-1}(M_o)] \cup [\mu_t^{n-1}(M_o) \times (R_{au} \cup R_{ad})]\}$.

Montrons qu'une décomposition de rang n

de : $.m-r \longrightarrow m + .r$

de tout radical $.m-r$ de $R_{au} \cup R_{ad}$, produit un radical .r appartenant à $\rho_t^{n-1}(M_o) \cup R_{au} \cup R_{ad}$.

1er cas : $.m-r \in R_{au}$

Alors il existe une molécule $u-v$ de M_{n-1}, telle que

au : $u-v \longrightarrow .u + .v$ et $.u = .m-r$.

$u-v$ est donc aussi de la forme $r-t$ avec $.t = .m-v$, et nous avons un amorçage :

au : $r-t \longrightarrow .r + .t$ avec $r-t \in M_{n-1}$.

Donc :

$.r \in R_{au}$.

2ème cas : $.m-r \in R_{ad}$

Alors il existe une molécule m' de $\rho_t^{n-1}(M_o)$ et un radical .u de $\rho_t^{n-1}(M_o) \cup R_{au} \cup R_{ad}$ tels que :

ad : $m' + .u \longrightarrow .m'-u$ et $.m'-u = .m-r$.

Si $.u = .r$, alors .r appartient à $\rho_t^{n-1}(M_o) \cup R_{au} \cup R_{ad}$.
Sinon, le groupe d'atomes r est contenu dans m', c'est-à-dire que m' est de la forme $r-t$. Par suite nous avons l'amorçage :

au : $r-t \longrightarrow .r + .t$

Comme $m' \in \mu_t^{n-1}(M_o)$, $.r \in \rho_t^{n-1}(M_o) \cup R_{au}$. □

Nous traiterons donc les processus simples dans l'ordre suivant :

- Les amorçages unimoléculaires.

Ceux-ci sont construits par une procédure *auens* qui a pour paramètres d'entrée *mininf* et *minsup* qui définissent l'ensemble de molécules à traiter, et pour paramètres mis à jour *lastr* et *lastp*, qui définissent l'indice du dernier radical et du dernier processus (les amorçages ne produisent pas de molécules).

procédure auens (in mininf, minsup : indxcnst ;
inout lastr : indxcnst ;
inout lastp : indxpr) ;

- Les additions.

Celles-ci sont construites par une procédure *adens*. L'ensemble des réactifs moléculaires est fixé par les paramètres en entrée *mininf* et *minsup*, celui des réactifs radicalaires par les paramètres *rininf* et *rinsup*. Les additions produisent uniquement des radicaux libres, dont la taille est limitée par *t*. Donc il faut un nouveau paramètre en entrée. Les paramètres mis à jour sont *lastr* et *lastp* (les additions ne produisent pas de molécules).

procédure adens (in t : entier ;
in mininf, minsup, rininf, rinsup : indxcnst ;
inout lastr : indxcnst ;
inout lastp : indxpr) ;

- Les décompositions.

La procédure *deens* traite les décompositions de l'ensemble des radicaux défini par les paramètres en entrée *rininf* et *rinsup*. Nous avons vu que les décompositions ne produisent pas de nouveaux radicaux libres : donc les paramètres mis à jour sont *lastm* et *lastp*.

```
procédure deens (in rininf, rinsup : indxcnst ;
                 inout lastm : indxcnst ;
                 inout lastp : indxpr) ;
```

- Les combinaisons.

Celles-ci sont traitées par la procédure *coens*. Elle a deux paramètres en entrée, *rininf* et *rinsup* qui définissent un ensemble de radicaux, et deux paramètres mis à jour, *lastm* et *lastp* (les combinaisons ne produisent pas de radicaux).

```
procédure coens (in rininf, rinsup : indxcnst ;
                 inout lastm : indxcnst ;
                 inout lastp : indxpr) ;
```

Au rang n, [*rininf, rinsup*] définit seulement les radicaux de R_n. Mais il faut aussi les combiner avec ceux de $\rho_t^{n-1}(M_n)$ définis par [*1, rininf-1*]

Nous donnons maintenant les algorithmes des procédures *mécaprim* et *mécagen*.

```
procédure mécaprim (in t : entier ;
                    in minsup : indxcnst ;
                    inout lastm, lastr : indxcnst ;
                    inout lastp : indxpr) ;
var x1, x2 : indxcnst ;
début
                              % amorçages unimoléculaires avec Mo %
  auens (1, minsup, lastr, lastp) ;
  x1 := 1 ; x2 := lastr ;     % [x1, x2] sont des radicaux de R1 %
  tantque x2 ⩾ x1 faire
                              % additions avec Mo et [x1, x2] %
    adens (t, 1, minsup, x1, x2, lastr, lastp) ;
                % [x2 + 1, lastr] sont de nouveaux radicaux de R1 %
    x1 := x2 + 1 ; x2 := lastr
  ftantque                    % [1, lastr] = R1 %
                              % décompositions avec R1 %
  deens (1, lastr, lastm, lastp) ;
                              % combinaisons avec [1, lastr] %
  coens (1, lastr, lastm, lastp) ;
  'construire les processus composés' ;
  imprimer (minsup + 1, lastm) ;     % molécules primaires %
  imprimer (1, routsup) ;            % radicaux primaires %
fin ; % mécaprim %
```



```
procédure mécagen (in t : entier ;
                   in mininf, minsup, rinsup : indxcnst ;
                   inout lastm, lastr : indxcnst ;
                   inout lastp : indxpr) ;
var x1, x2 : indxcnst ;
début
                              % amorçages unimoléculaires avec M_{n-1} %
  auens (mininf, minsup, lastr, lastp) ;
                              % additions avec M_{n-1} et ρ_t^{n-1}(M_o) %
  adens (t, mininf, minsup, 1, rinsup, lastr, lastp) ;
                              % additions avec μ_t^{n-1}(M_o) et R_n %
  x1 := rinsup + 1 ; x2 := lastr ;
                              % [x1, x2] sont des radicaux de R_n %
  tantque x2 ⩾ x1 faire
                              % additions avec μ_t^{n-1}(M_o) et [x1, x2] %
    x1 := x2 + 1 ; x2 := lastr ;
  ftantque ;                  % [rinsup + 1, lastr] définit R_n %
                              % décomposition avec R_n %
    deens (rinsup + 1, lastr, lastm, lastp) ;
                              % combinaisons avec R_n %
    coens (rinsup + 1, lastr, lastm, lastp) ;
    'construire les processus composés' ;
    imprimer (minsup + 1, lastm) ;          % molécules de rang n %
    imprimer (rinsup + 1, lastr) ;          % radicaux de rang n %
fin ; % mécagen %
```

Commentaires :

1) La procédure *adens* est appelée dans une boucle "tantque", car nous avons vu qu'un processus d'addition pouvait produire des nouveaux radicaux, et que ces radicaux étaient à nouveau candidats à une addition (cf. exemple 2, § 3.2.). Les additions ne sont faites que si la taille du radical produit est inférieure à t et nous avons montré (proposition 1, § 3.2.) que cette condition implique que l'ensemble des produits de rang n est alors de cardinal fini : ceci assure la terminaison de l'algorithme.

2) Les processus composés sont construits à partir des processus simples de rang inférieur. Nous n'explicitons pas l'algorithme de cette recherche car elle dépend de l'implantation du programme, et qu'elle n'a pas d'influence sur la recherche des processus simples, ceux-ci étant uniquement définis par l'ensemble des réactifs.

3) Les appels de chaque procédure *auens*, *adens*, *deens*, *coens* s'appliquent chaque fois à un ensemble de réactifs disjoints. En conséquence les lots de processus construits aux différents appels sont distincts.

4) Dans *mécaprim*, après l'appel de *auens*, on peut avoir *lastr = 0* et *lastp = 0*, ce qui correspond à une génération vide. Dans ce cas *adens* n'est jamais appelé.

Enfin, nous explicitons les algorithmes des procédures *auens*, *adens*, *deens*, *coens*.

La procédure *auens* traite les amorçages unimoléculaires avec les molécules de *ensmol*, d'indices compris entre *mininf* et *minsup*. Elle appelle une procédure *auelt* qui cherche tous les amorçages possibles avec une molécule et les imprime ; son interface est :

```
procédure auelt (in m : molécule ; inout last r : indxcnst ;
                 inout lastp : indxpr) ;
procédure auens (in mininf, minsup : indxcnst ;
                 inout lastr : indxcnst ; lastp : indxpr) ;
var i : indxcnst ;
début
  pour i := mininf à minsup faire
    auelt (ensmol[i], lastr, lastp) ;
  fpour ;
fin ; % auens %
```


La procédure *adens* traite les additions des radicaux de *ensrad*, d'indices compris entre *rininf* et *rinsup*, sur les molécules de *ensmol*, d'indices compris entre *mininf* et *minsup*. Il faut que la taille du radical produit soit inférieure à t. Nous avons donc une

```
fonction taille (in c : constituant) : entier ;
```

qui donne le nombre d'atomes d'un constituant moléculaire ou radicalaire (le type *constituant* est à définir). La procédure *adens* appelle une procédure *adelt* qui cherche tous les processus d'addition d'un radical sur une molécule, et les imprime.

```
procédure adelt (in m : molécule ; in r : radical ;
                 inout lastr : indxcnst ; inout lastp : indxpr) ;
procédure adens (in t : entier ;
                 in mininf, minsup, rininf, rinsup : indxcnst ;
                 inout lastr : indxcnst ; inout lastp : indxpr) ;
var i, j : indxcnst ;
début
  pour i := mininf à minsup faire
    pour j := rininf à rinsup faire
      si taille (ensmol[i]) + taille (ensrad[j]) < t
        alors adelt (ensmol[i], ensrad[j], lastr, lastp)
        sinon % on ne considère pas le processus %
      fsi ;
    fpour ;
  fpour ;
fin ; % adens %
```

La procédure *deens* traite les décompositions des radicaux de *ensrad* compris entre *rininf* et *rinsup*. Pour chaque radical, elle appelle une procédure *deelt* qui cherche toutes les décompositions et les imprime. Seul l'indice de la dernière molécule produite est repéré.

```
procédure deelt (in r : radical ;
                 inout lastm : indxcnst ; inout lastp : indxpr) ;
procédure deens (in rininf, rinsup : indxcnst ;
                 inout lastm : indxcnst ; inout lastp : indxpr) ;
```

```
var i : indxcnst ;
début
  pour i := rininf à rinsup faire
    deelt (ensrad[i], lastm, lastp) ;
  fpour ;
fin ; % deens %
```

La procédure *coens* doit traiter la combinaison des radicaux de $\{ensrad[i],\ i \in [1,\ rininf-1]\}$ avec ceux de $\{ensrad[i],\ i \in [rininf,\ rinsup]\}$ puis ces derniers radicaux entre eux. La

```
procédure coelt (in r1, r2 : radical ;
                 inout lastm : indxcnst ; inout lastp : indxpr) ;
```

traite la combinaison de *r1* et *r2*, et l'imprime.

```
procédure coens (in rininf, rinsup : indxcnst ;
                 inout lastm : indxcnst ; inout lastp : indxpr) ;
var i, j : indxcnst ;
début
  pour i := 1 à rininf-1 faire
    pour j := rininf à rinsup faire
      coelt (ensrad[i], ensrad[j], lastr, lastp) ;
    fpour ;
  fpour ;
  pour i := rininf à rinsup faire
    pour j := i à rinsup faire
      coelt (ensrad)[i], ensrad[j], lastr, lastp) ;
    fpour ;
  fpour ;
fin ; % coens %
```

Les algorithmes des procédures *auelt*, *adelt*, *deelt* et *coelt* dépendent de la structure de données choisie pour les molécules et les radicaux libres. Le choix de cette structure est le but du chapitre III et nous donnerons ces algorithmes dans le chapitre IV.

CHAPITRE III

Notation des constituants et représentation interne canonique

Dans ce chapitre nous définissons d'une part une notation pour les constituants utilisée pour la lecture des réactifs d'une réaction et pour l'impression de son mécanisme réactionnel, d'autre part une représentation interne canonique des constituants adaptée à l'application des modèles de processus élémentaires.

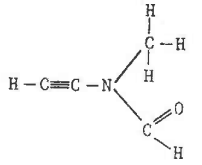
Les constituants sont vus comme des graphes ; on pourra se référer à [BERGE 1971] pour les définitions et résultats de cette théorie.

1. GRAPHE D'UN CONSTITUANT.

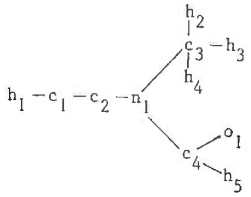
La formule développée d'un constituant, se présente sous la forme d'un graphe connexe, non orienté, sans boucle, dont les noeuds, en nombre fini, sont associés aux atomes, et les arêtes aux liaisons. Chaque noeud est étiqueté par le symbole chimique de l'atome, assorti d'un point si l'atome porte un électron célibataire et chaque liaison est étiquetée par sa nature (simple, double ou triple). Il convient de noter qu'entre deux noeuds, il y a au plus une arête, bien que les liaisons doubles ou triples soient représentées par des traits doublés ou triplés. Comme nous avons éliminé tous les constituants cycliques le graphe est sans cycle.

Exemple :

La molécule dont la formule développée est la suivante :



est associé au graphe :

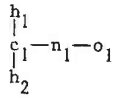


Les noeuds c_i, i de 1 à 4, portent l'étiquette C, les noeuds h_i, i de 1 à 5, l'étiquette H, le noeud o_1 l'étiquette O, et le noeud n_1 l'étiquette N.

L'arête $\{c_1, c_2\}$ porte l'étiquette triple, l'arête $\{c_4, o_1\}$ l'étiquette double ; toutes les autres portent l'étiquette simple.

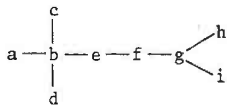
De la même façon, le radical libre, de formule développée :

est associé au graphe :



Mais l'étiquette de c_1 est .C

Un graphe non orienté peut être considéré comme un graphe orienté symétrique. Ainsi la représentation orientée, du graphe non orienté suivant :



est :

Utilisant les notations et conventions habituelles, un graphe G est défini par un couple (X, Γ) où X est l'ensemble des noeuds et Γ une relation binaire sur X qui définit un sous-ensemble de $X \times X$, appelé ensemble des arcs de G.

L'expression booléenne $x \Gamma y$ est vraie si (x, y) est un arc de G.

Pour éviter toute ambiguïté sur la notion de cycle utilisée, nous donnons les deux définitions suivantes [MARCHAND 1979].

Définition 1 :

Une chaîne d'un graphe (X, Γ) est une suite de noeuds $[x_o, x_1, \ldots, x_n]$ tels que

$$x_{i-1} \Gamma x_i \quad \text{ou} \quad x_i \Gamma x_{i-1}, \quad \text{pour } i \text{ de } 1 \text{ à } n .$$

Définition 2 :

Une chaîne $[x_o, x_1, \ldots, x_n]$ est un cycle si $x_o = x_n$ et si $1 \leqslant i \leqslant n-1 \Rightarrow x_{i-1} \neq x_{i+1}$

Remarques :

1) Une boucle est un cycle.

2) La deuxième condition imposée dans la définition entraîne que $[x_1, x_2, x_1]$ n'est jamais un cycle, sans cette condition tout graphe (X, Γ) admettant au moins un arc admet des cycles.

Avec cette définition, nous pouvons dire que le graphe (X, Γ) associé à un constituant est un graphe symétrique (car Γ l'est), connexe, et sans cycle. Ce type de graphe est couramment appelé arbre.

Nous terminons ce paragraphe, en donnant deux propositions utiles dans la suite.

Proposition 1 :

Soit $G = (X, \Gamma)$ un arbre. Pour tout point r de X, il existe un et un seul graphe partiel de G qui soit une arborescence de racine r. Appelons G_r cette arborescence et Γ_r la relation qui la définit :

$$G_r = (X, \Gamma_r)$$

Démonstration :

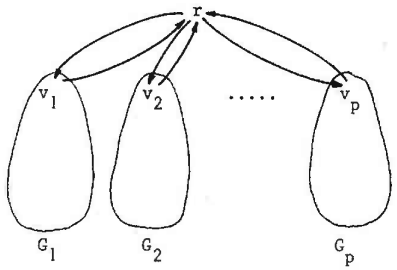
Raisonnons par récurrence sur $|X|$. Considérons d'abord le cas d'un arbre G dont l'ensemble des noeuds est réduit à $\{r\}$. Alors G n'a pas d'arcs et c'est une arborescence.

Supposons que pour tout arbre $G = (X, \Gamma)$ ayant au plus n noeuds et pour tout choix d'un noeud r de X, il existe un et un seul graphe partiel de G qui soit une arborescence de racine r.

Considérons maintenant un arbre $G = (X, \Gamma)$ ayant $n + 1$ noeuds et soit r un élément de X.

Soient :

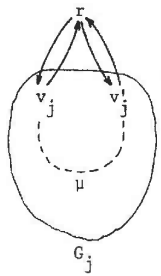
- $V = \{v_1, v_2, \ldots, v_p\}$ l'ensemble des voisins de r
- G' le sous-graphe de G engendré par $\{X - r\}$;
- pour i de 1 à p, soit $G_i = (X_i, \Gamma_i)$ la composante connexe de G' contenant v_i.



Montrons que : $\forall i$ de 1 à p, $V \cap X_i = \{v_i\}$.

- Chaque X_i contient v_i, par définition.

- Supposons qu'il existe j tel que $V \cap X_j$ contienne deux noeuds distincts v_j et v'_j. G_j étant connexe, il existe une chaîne élémentaire $\mu = [v_j, \ldots, v'_j]$ de G_j reliant les deux noeuds. Construisons la chaîne μ' de G en ajoutant les arcs (r, v_j) et (v'_j, r) respectivement au début et à la fin de la chaîne μ. ((r, v_j) et (v'_j, r) sont des arcs de G car v_j et v'_j sont des voisins de r dans G et G est symétrique).



La chaîne μ' ainsi construite est un cycle de G ce qui est impossible.

Donc : $\forall i$ de 1 à p, $V \cap X_i = \{v_i\}$.

Chaque composante connexe G_i est un arbre ayant au plus n noeuds. Ayant fait choix du noeud v_i, nous pouvons construire un graphe partiel unique $G_{i_{v_i}} = (X, \Gamma_{i_{v_i}})$ qui soit une arborescence de racine v_i.

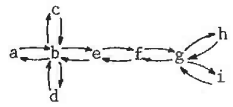
Soit Γ_r la relation définie par :

- $\forall i \in [1, p]$, $r \Gamma_r v_i$
- $\forall i \in [1, p]$, $\forall (x, y) \in X_i^2$, $x \Gamma_r y = x \Gamma_{i_{v_i}} y$.

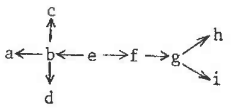
Il est facile de vérifier que $G_r = (X, \Gamma_r)$ est un graphe partiel de G, et que c'est une arborescence de racine r. De plus la construction de G_r montre qu'elle est unique. □

Exemple :

Soit G l'arbre suivant :



L'arborescence G_e de racine e déduite de G est :



Proposition 2 :

Soient $G = (X, \Gamma)$ un arbre, r un point de X, $G_r = (X, \Gamma_r)$ l'arborescence de racine r déduite de G. Alors :

$$\Gamma_r \cup \Gamma_r^{-1} = \Gamma$$

Démonstration :

Γ_r est contenue dans Γ qui est symétrique ; donc :

$$\Gamma_r \cup \Gamma_r^{-1} \subset \Gamma$$

Pour montrer la réciproque nous démontrons sa contraposée. Supposons qu'il existe x et y dans X tel que

$$\rceil(x \Gamma_r y) \quad \text{et} \quad \rceil(y \Gamma_r x)$$

1er cas : $x = y$.

Comme G est un arbre, nous avons forcément $\rceil(x \Gamma y)$.

2ème cas : $x \neq y$.

Nous ne pouvons avoir à la fois $x = r$ et $y = r$ et, quitte à échanger les noms de x et de y, nous supposons $x \neq r$.

Soient $[r, x_o, x_1, \ldots, x_n = x]$, $n \geqslant 0$, l'unique chemin de r à x, et $[r = y_o, y_1, \ldots, y_p = y]$, $p \geqslant 0$, l'unique chemin de r à y dans G_r.

Si $n = 0$ et $p = 0$ alors $r \Gamma_r x$ et $y = r$ ce qui est impossible. Donc nous avons : $n \neq 0$ ou $p \neq 0$.

D'autre part :

$$\rceil(y \Gamma_r x) \quad \text{et} \quad n \neq 0 \implies x_{n-1} \neq y$$

$$\rceil(x \Gamma_r y) \quad \text{et} \quad p \neq 0 \implies y_{p-1} \neq x$$

Comme G_r est un sous-graphe partiel de G, Γ_r est contenue dans Γ et la suite de points $(x_n = x, x_{n-1}, \ldots, x_o, r = y_o, y_1, \ldots, y_p = y)$ est une chaîne de G telle que $x_{n-1} \neq x$ ou $y_{p-1} \neq y$.

Puisque G est sans cycle nous ne pouvons avoir $x \Gamma y$.

Donc :

$$\forall (x, y) \in X^2, \ \rceil(x \Gamma_r y) \quad \text{et} \quad \rceil(y \Gamma_r x) \implies \rceil(x \Gamma y)$$

ce qui est équivalent à :

$$\forall (x, y) \in X^2, \ x \Gamma y \implies x \Gamma_r \cup \Gamma_r^{-1} y$$

□

2. NOTATION DES CONSTITUANTS.

2.1. Introduction.

Nous devons disposer d'une notation chimique des constituants, utilisable dans un contexte informatique qui permette à un chimiste d'entrer la liste des réactifs et de lire le mécanisme construit par l'application. Cette notation doit décrire complètement la structure d'une molécule, car pour appliquer un modèle de processus élémentaire, il faut connaître toutes les liaisons qui existent entre les atomes. Cependant, seule l'existence ou la non existence d'une liaison simple, double ou triple nous intéresse, mais pas leur longueur ni leur angle.

Il nous fallait répondre aux mêmes contraintes que celles fixées par le langage du compilateur chimique (cf. chapitre I § 3.1.) c'est-à-dire définir une notation linéaire, compatible avec les périphériques classiques, proche des notations utilisées habituellement par les chimistes. Mais la notation du compilateur ne permet qu'une identification des constituants chimiques, et reste ambiguë. Par exemple, l'expression C5H12 est acceptée par le compilateur, mais peut correspondre à plusieurs isomères (la spécification par du texte, complétant une formule brute pour en distinguer les isomères, comme dans "NEO" C5H12, ne fait l'objet d'aucune analyse sémantique permettant de restituer la structure de la molécule). En revanche, une expression telle que C(CH3)4, également acceptée par le compilateur, décrit la structure de la molécule, mais ce fait n'y est pas exploité. Nous avons donc raffiné les règles d'écriture des constituants établies pour le compilateur, pour définir un langage de description des constituants. Comme nous l'avons indiqué dans le chapitre II, nous nous bornerons aux molécules et aux (mono-) radicaux libres covalents, sans cycle et dans lesquels toutes les occurrences d'un même symbole atomique ont la même valence.

2.2. Notation.

Nous présentons successivement la notation des éléments de base, des molécules et des radicaux libres.

2.2.1. Notation des éléments de base.

Les éléments de base qui apparaissent dans les composés envisagés sont les symboles atomiques, les liaisons et les électrons célibataires.

2.2.1.1. Les symboles atomiques.

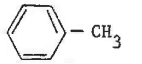
Nous avons repris les conventions faites pour le langage du compilateur.

- Les symboles atomiques d'une seule lettre sont décrits par cette lettre : C pour le carbone, H pour l'hydrogène, etc...

- Les symboles atomiques de deux lettres sont écrits entre apostrophes : 'CL' pour le chlore, 'CO' pour le cobalt, etc...

- Il est possible d'attribuer à une lettre quelconque, X, par exemple, ou à une suite de lettres et de chiffres entre apostrophes, 'ATOME', 'A1', 'A2' par exemple, la propriété usuellement réservée aux atomes, à savoir celle de ne pas être modifiés par réaction chimique.

Cette possibilité est utile pour les chimistes lorsqu'ils veulent étudier des mécanismes formels généraux. Elle permet aussi d'écrire des composés cycliques, moyennant quelques conventions propres à un utilisateur. Par exemple, si 'PHI' représente le noyau benzénique, avec une valence libre, le toluène de formule développée



peut se noter 'PHI'/CH3

2.2.1.2. Les liaisons.

De même que dans le compilateur, les liaisons simples, doubles et triples sont notées respectivement /, //, ///.

2.2.1.3. Les électrons célibataires.

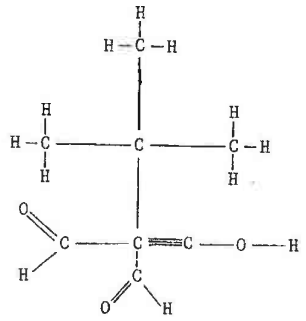
Un électron célibataire est noté simplement par un point, écrit devant le symbole chimique de l'atome qui le porte.

2.2.2. Notation des molécules.

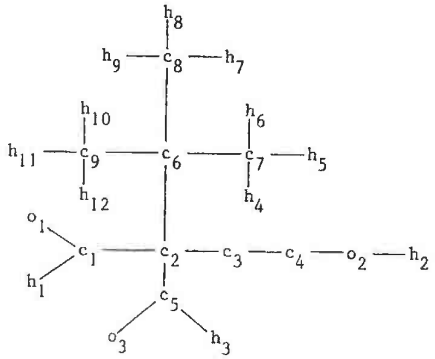
2.2.2.1. Principe général.

La notation d'une molécule s'élabore à partir de l'arbre étiqueté, dérivé de sa formule développée. Nous allons en exposer le principe général sur un exemple.

Soit la molécule ayant la formule développée suivante :



L'arbre G = (X, Γ), où X est l'ensemble des noeuds et E l'ensemble des arêtes est le suivant :

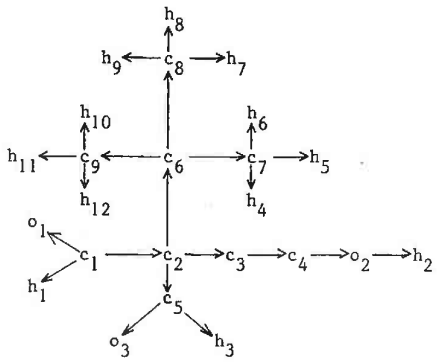


Les sommets c_i, i de 1 à 9, portent l'étiquette C ;
les sommets h_i, i de 1 à 12, portent l'étiquette H ;
les sommets o_i, i de 1 à 13, portent l'étiquette O.
L'arête $\{c_3, c_4\}$ est étiquetée par une liaison triple (///) ;
les arêtes $\{o_1, c_1\}$ et $\{c_3, c_5\}$ sont étiquetées par une liaison double (//) ;
les autres arêtes sont étiquetées par des liaisons simples (/).

Etape 1 :

Dans l'arbre G, nous choisissons un noeud quelconque, appelé foyer. La proposition 1 indique qu'alors il existe un et un seul graphe partiel de G qui soit une arborescence, dont la racine est le noeud choisi.

Choisissons, par exemple, c_1 comme foyer, mais tout autre noeud peut convenir. L'arborescence G_{c_1} de racine c_1 déduite de G est la suivante :



Etape 2 :

Dans l'arborescence G_{c_1}, nous choisissons un chemin (élémentaire) d'extrêmité initiale c_1, appelé chemin principal. Prenons, par exemple, le chemin $[c_1, c_2, c_3]$. Nous écrivons les étiquettes des noeuds et des arêtes dans l'ordre d'apparition dans le chemin ; nous obtenons l'expression N_1 :

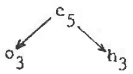
$$N_1 = C/C/C$$

Etape 3 et suivantes :

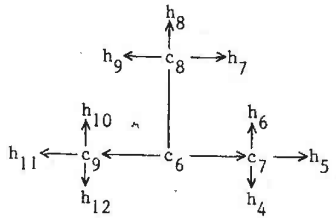
Soit $A = \{c_1, c_2, c_3\}$ l'ensemble des noeuds notés dans N_1 ; le sous-graphe de G_{c_1}, engendré par X−A, a cinq composantes connexes, qui sont des arborescences ; leur racine est leur seul noeud lié à un élément de A dans G_{c_1}.

o_1 la racine o_1 est liée à c_1 ;

h_1 la racine h_1 est liée à c_1 ;



la racine c_5 est liée à c_2 ;



la racine c_6 est liée à c_2 ;

$c_4 \longrightarrow o_2 \longrightarrow h_2$ la racine c_4 est liée à c_3.

Remarque :

Dans le vocabulaire chimique, ces composantes connexes sont appelées substituants. Par exemple l'arborescence $c_4 \longrightarrow o_2 \longrightarrow h_2$ associée à la formule développée ≡ C — O — H est le substituant de l'atome C (numéroté 3), et on dit que C est substitué.

Dans chacune de ces composantes, nous choisissons à nouveau un chemin commençant par la racine, appelé chemin secondaire. Nous prendrons par exemple les chemins $[o_1]$, $[h_1]$, $[c_5, h_3]$, $[c_6, c_7]$, $[c_4, o_2, h_2]$. Pour chaque chemin, nous écrivons, entre parenthèses, l'étiquette de l'arête qui lie la racine à un noeud de A, puis les étiquettes des noeuds et des arêtes, comme précédemment. Nous obtenons les expressions suivantes :

(//O), (/H), (/C/H), (/C/C), (///C/O/H)

Ces expressions sont insérées dans N_1, après l'atome substitué. Si un atome a plusieurs substituants on écrit les expressions correspondantes les unes à la suite des autres dans un ordre arbitraire ; la notation N_1 devient :

N_2 = C(//O)(/H)
/C(/C/H)(/C/C)
/C(///C/O/H)

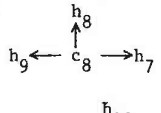
Nous réitérons l'étape 3 jusqu'à ce que tous les noeuds du graphe soient notés.

Etape 4 :

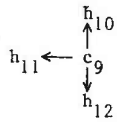
$A = \{c_1, c_2, c_3, o_1, h_1, c_5, h_3, c_6, c_7, c_4, o_2, h_2\}$

Les composantes connexes du sous-graphe engendré par X—A sont :

o_3 la racine o_3 est liée à c_5 ;



la racine c_8 est liée à c_6 ;



la racine c_9 est liée à c_6 ;

h_4 la racine h_4 est liée à c_7 ;

h_5 la racine h_5 est liée à c_7 ;

h_6 la racine h_6 est liée à c_7.

Les chemins secondaires et leurs expressions associées sont :

$[o_3]$	(//O)
$[c_8, h_7]$	(/C/H)
$[c_9]$	(/C)
$[h_4]$	(/H)
$[h_5]$	(/H)
$[h_6]$	(/H)

La notation N_2 devient :

N_3 = C(//O)(/H)
/C(/C(//O)(/H)
(/C(/C/H)(/C)/C(/H)(/H)(/H))
/C(///C/O/H)

Etape 5 :

$A = \{c_1, c_2, c_3, o_1, h_1, c_5, h_3, c_6, c_7, c_4, o_2, h_2,$
$o_3, c_8, h_7, c_9, h_4, h_5, h_6\}$

Les composantes connexes du sous-graphe engendré par X—A sont :

h_8 la racine h_8 est liée à c_8 ;

h_9 la racine h_9 est liée à c_8 ;

h_{10} la racine h_{10} est liée à c_9 ;

h_{11} la racine h_{11} est liée à c_9 ;

h_{12} la racine h_{12} est liée à c_9 ;

La notation N_3 devient N_4 :

N_4 = C(//O)(/H)
/C(/C(//O)(/H))
(/C(/C(/H)(/H)/H)(/C(/H)(/H)(/H))/C(/H)(/H)(/H))
/C(///C/O/H)

2.2.2.2. Règles de simplification.

a) Suppression des liaisons simples.

On peut supprimer toute liaison simple qui suit une parenthèse ouvrante.

Avec cette règle N_4 devient :

N_5 = C(//O)(H)
/C(C(//O)(H))
(C(C(H)(H)/H)(C(H)(H)(H))/C(H)(H)(H))
/C(///C/O/H)

b) Factorisation des expressions.

Si un atome a plusieurs substituants identiques on peut écrire l'expression associée une seule fois et écrire le nombre de substituants après la parenthèse fermante.

N_5 devient :

N_6 = C(//O)(H)
/C(C(//O)(H))
(C(C(H)2/H)(C(H)3)/C(H)3)
/C(///C/O/H)

c) Suppression des parenthèses.

Lorsqu'une expression entre parenthèses ne contient que le symbole chimique d'un atome, on peut supprimer les parenthèses :

N_6 devient :

N_7 = C(//O)H
/C(C(//O)H)
(C(CH2/H)(CH3)/CH3)
/C(///C/O/H)

L'utilisation de ces règles est libre : une notation telle que C(/H)(/H)(/H)(/H) est tout aussi valable que CH_4.

2.2.2.3. Les molécules ayant des symétries.

Certaines molécules présentent des symétries. Les éléments de symétrie sont multiples, et nous ne parlerons que des symétries par rapport à un atome comme dans le propane ($CH_3-CH_2-CH_3$), et à une liaison comme dans l'azote ($N\equiv N$) ou l'éthane (CH_3-CH_3).

La notation précédemment définie offre la possibilité de mettre en évidence un atome centre de symétrie : il suffit de le choisir comme foyer. Par exemple nous pouvons écrire CH2(CH3)2 pour le propane. Ceci n'est plus vrai si l'élément de symétrie est une liaison, et nous avons ajouté une règle d'écriture supplémentaire.

Soient $G = (x, \Gamma)$ l'arbre associé à une molécule ayant une liaison centre de symétrie, et $x_1 \Gamma x_2$ l'arête associée à cette liaison. Le graphe partiel $G' = (X, \Gamma')$ obtenu à partir de G en supprimant l'arête $x_1 \Gamma x_2$ a deux composantes connexes identiques. Nous cherchons une notation N associée à l'une de ces composantes dont le foyer est x_1 (ou x_2). La notation de la molécule est alors obtenue en écrivant l'étiquette de l'arête $x_1 \Gamma x_2$, puis l'expression de N entre parenthèses, et enfin l'entier 2 qui indique qu'une factorisation a été faite. Nous appellerons "forme symétrique" ce type de notation, et dirons que $\{x_1, x_2\}$ est un foyer double. Par analogie nous appellerons "forme non symétrique" la notation générale bien qu'elle couvre le cas d'un atome de symétrie.

Exemples :

- l'hydrogène, H—H, est noté N_1 = /(H)2 ;
- l'azote, $N\equiv N$, est noté N_2 = ///(N)2 ;
- l'éthane, CH_3-CH_3, est noté N_3 = /(CH3)2 ;

Les règles de simplification permettent de supprimer les parenthèses dans N_1 et N_2 :

N_1 = /H2 et N_2 = ///N2

Nous ajoutons une nouvelle règle :

dans une forme symétrique, on peut supprimer la liaison de symétrie si elle est simple.

Avec cette règle, nous obtenons :

N_1 = H2 et N_3 = (CH3)2

Là encore, l'utilisation d'une forme symétrique est libre et l'éthane, par exemple, peut être noté CH3/CH3. Mais nous verrons dans le paragraphe 3 de ce chapitre que les éléments de symétrie, atome ou liaison, sont recherchés systématiquement, à partir de n'importe quelle notation d'entrée.

2.2.3. Notation des radicaux libres.

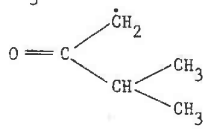
Les radicaux que nous considérons ont un atome, et un seul, porteur d'un électron célibataire. Dans l'arbre associé, cet atome, dit pointé, est étiqueté par son symbole chimique précédé d'un point.

La notation des radicaux libres s'élabore en suivant le même principe que pour les molécules. Mais on impose que l'atome pointé soit pris comme foyer, ce qui est naturel, car il est distingué. En conséquence, la notation d'un radical libre commence toujours par un point.

Exemples :

$\dot{H}$ est noté .H

$\dot{C}H_3$ est noté .CH3



est noté .CH2/C(//O)/CH(CH3)2

Remarquons que nous ne pouvons pas avoir de forme symétrique pour la notation d'un radical libre, car l'atome pointé détruit toute symétrie éventuelle par rapport à une liaison.

2.2.4. Conclusion.

Les règles établies pour la notation d'un constituant laissent une grande liberté au chimiste :

- le choix du foyer dans une molécule, des chemins principal et secondaires est totalement libre ;

- l'ordre d'écriture des substituants d'un même atome est arbitraire ;

- l'application des règles de simplification n'est pas imposée ;

- la mise en évidence d'un élément de symétrie, atome ou liaison, est une possibilité, non une obligation.

Tout ceci confère à cette notation une certaine souplesse. Mais en contrepartie, ce n'est évidemment pas une notation canonique. Ce problème est traité au paragraphe 3 de ce chapitre, mais auparavant nous donnons la grammaire formelle des constituants.

3. GRAMMAIRE DES CONSTITUANTS.

3.1. Grammaire formelle.

Cette grammaire est décrite selon la forme normale de BAKUS (B.N.F.) (le lecteur pourra consulter [GROSS 1967]).

3.1.1. Le vocabulaire terminal.

Les mots terminaux nécessaires à la notation d'un constituant sont :

- les atomes,
- les liaisons,
- les entiers,
- les parenthèses ouvrante et fermante,
- le point radicalaire.

L'entier "2" joue un rôle particulier dans la notation des molécules ayant une liaison centre de symétrie. Nous rajoutons le mot terminal "deux", pour simplifier la définition de la grammaire des constituants, bien que nous introduisons, au niveau lexicographique, une indétermination entre "2" reconnu comme un entier, utilisé pour factoriser deux substituants identiques, et "2" qui indique la factorisation particulière faite dans les molécules symétriques. Cette indétermination n'est que théorique, et est très courante dans les analyseurs lexicographiques. En pratique cela ne pose pas de problème particulier.

Dans les règles d'écriture suivantes, le symbole "::=" est l'opérateur de dérivation, la barre verticale, "|", indique un choix entre deux alternatives, et "A" désigne le mot vide.

```
    atome ::= lettre | ' lettre suitatome '
suitatome ::= lettre suitatome | chiffre suitatome | A
  liaison ::= / | // | ///
   entier ::= chiffre suitentier
suitentier ::= chiffre suitentier | 0 suitentier | A
       po ::= (
       pf ::= )
    point ::= .
     deux ::= 2
   lettre ::= A | B | ... | Z
  chiffre ::= 1 | ... | 9
```


Remarques :

1) Le texte est lu sur une image carte dont le nombre de caractères est défini par une constante *maxcarte* et chaque mot terminal doit être contenu sur une même carte. Il s'ensuit qu'un atome défini par une chaîne de caractères entre deux apostrophes contient au plus *maxcarte* caractères, y compris les deux apostrophes.

2) Les caractères espaces ne sont pas admis à l'intérieur d'un mot terminal. Mais entre deux mots terminaux le nombre d'espaces est quelconque et sans signification.

3) L'identité de deux symboles atomiques est une égalité caractère par caractère. En conséquence les deux atomes X et 'X', par exemple, sont distincts.

3.1.2. Le vocabulaire non terminal.

Le tableau ci-dessous indique le nom et la signification des mots non terminaux.

nom	signification
<constituant>	constituant chimique moléculaire ou radicalaire
<molécule>	molécule décrite sous forme non symétrique
<radicallibre>	radical libre
<molsym>	molécule décrite sous forme symétrique
<groupe>	ensemble d'un atome et de ses substituants ; un <groupe> peut décrire une molécule sous forme non symétrique
<listesubst>	liste de substituants
<subst>	substituant
<L>	liaison implicite ou explicite
<M>	coefficient multiplicateur implicite ou explicite

3.1.3. Axiome.

L'axiome de la grammaire est <constituant>.

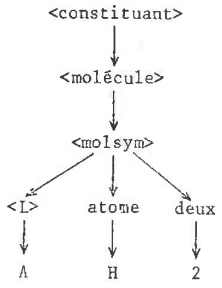
3.1.4. Règles de production.

Un constituant est une molécule ou un radical libre, ce qui nous donne les règles de production suivantes :

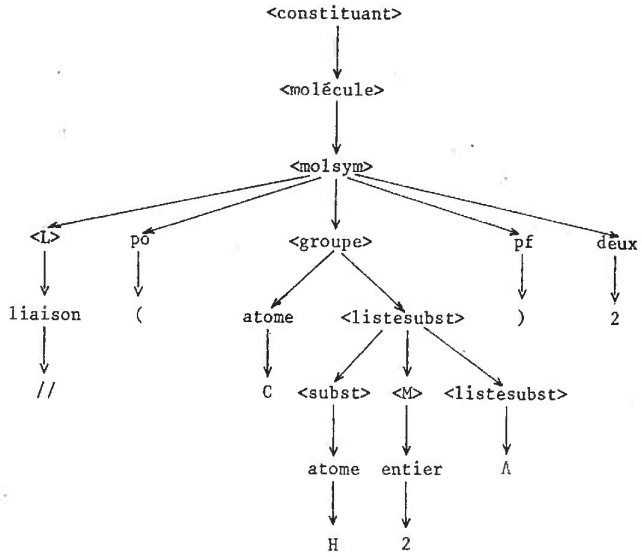
```
<constituant> ::= <molécule> | <radicallibre>
   <molécule> ::= <groupe> | <molsym>
     <groupe> ::= atome <listesubst>
     <molsym> ::= <L> atome deux
                | <L> po <groupe> pf deux
<radicallibre> ::= point <groupe>
  <listesubst> ::= <subst> <M> <listesubst>
                | liaison <groupe>
                | Λ
      <subst> ::= atome
                | po <L> <groupe> pf
          <L> ::= liaison | Λ
          <M> ::= entier | Λ
```

Exemples :

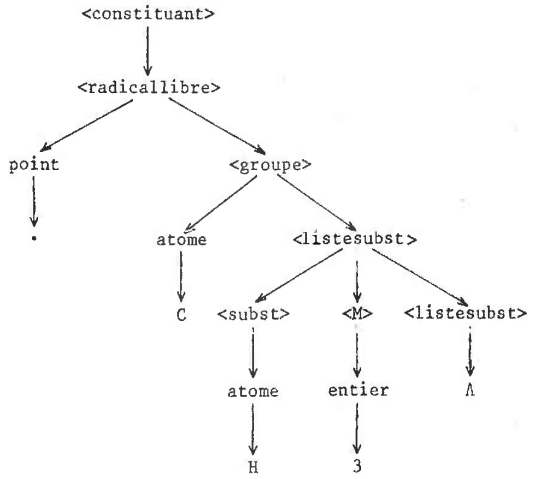
1) l'arbre syntaxique de "H2" est le suivant :



2) Arbre syntaxique de "//(CH2)2" :

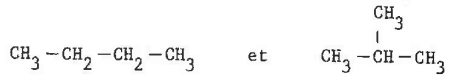


3) Arbre syntaxique de ".CH3" :



Commentaires sur la grammaire :

1) Un atome produit par le non terminal <groupe> n'est jamais suivi d'un entier : cela permet d'éliminer, au niveau syntaxique, les notations telles que C4H10, ou CH3/C2H4/CH3, qui sont amgibuës, car elles peuvent définir, chacune, les deux isomères suivants :



Les écritures acceptées sont, par exemple :

CH3/CH2/CH2/CH3 et CH3/CH(CH3)2

Cette restriction syntaxique est donc nécessaire, mais elle entraîne l'élimination des notations telles que C2H6, C3H9, qui ne sont pas ambiguës pour un chimiste.

2) Le non terminal <L> correspond à la règle de simplification 2.2.2.2.a s'il est produit par <subst> et à la règle propre aux formes symétriques (cf. § 2.2.2.3.), s'il est produit par <molsym> ; il désigne une liaison explicite, si <L> produit le terminal liaison, ou implicite, si <L> produit le mot vide Λ. Dans ce dernier cas, et d'après la définition de la notation des constituants, on sait qu'une liaison simple existe.

3) Le non terminal <M> correspond à la règle de simplification 2.2.2.2.b ; s'il produit le mot vide Λ, il faut restituer un entier de valeur 1.

4) La règle de production :

<subst> ::= atome

définit un substituant formé d'un seul atome. Sa liaison avec l'atome substitué est simple et est toujours implicite. Ceci correspond à la règle de simplification 2.2.2.2.c.

5) L'application des seules règles :

<molécule> ::= <groupe>
<groupe> ::= atome <listesubst>
<listesubst> ::= liaison <groupe> | Λ

avec l'axiome <molécule>, correspond au développement d'un chemin principal : une phrase de ce langage est de la forme $a_o\ l_1\ a_1\ l_2\ a_2 \cdots l_n\ a_n$

où les a_i, i de 0 à n, sont des atomes et, les l_i, i de 1 à n, des liaisons.

De la même façon, l'application des seules règles :

<subst> ::= po <L> <groupe> pf
<groupe> ::= atome <listesubst>
<listesubst> ::= liaison <groupe> | Λ

avec l'axiome <subst> est le développement d'un chemin secondaire : une phrase de ce langage est de la forme : $(l_1\ a_1\ l_2\ a_2 \cdots l_n\ a_n)$, la liaison l_1 pouvant être implicite.

Dans ces deux types de phrases, les règles :

<listesubst> ::= <subst> <M> <listesubst> | Λ,
<listesubst> étant initialement produit par :
<groupe> ::= atome <listesubst>,

permettent d'insérer des substituants après l'atome produit par cette dernière règle.

6) Considérons la phrase engendrée par l'axiome <groupe> et les règles :

<groupe> ::= atome <listesubst>
<listesubst> ::= <subst> <M> <listesubst>
| liaison <groupe>
| Λ

La sous-expression "liaison <groupe>" est un groupe d'atomes liés à l'atome produit par l'axiome. Elle décrit donc aussi un substituant de cet atome, et c'est pour cette raison que nous avons choisi de la produire à partir du non terminal <listesubst>, et non pas à partir de <groupe>, comme dans la grammaire (équivalente) suivante :

<groupe> ::= atome <listesubst> <suitegroupe>
<suitegroupe> ::= liaison <groupe>
| Λ
<listesubst> ::= <subst> <M> <listesubst>
| Λ

3.2. Sémantique de la grammaire.

La sémantique de la grammaire est exprimée au moyen d'attributs [KNUTH 1971a] [LORHO 1978]. Le but de cette sémantique est de construire une représentation interne pour un constituant moléculaire ou radicalaire reconnu par l'analyse syntaxique. Nous définissons :

- les objets de base associés aux mots terminaux ;
- les objets composés associés aux mots non terminaux ;
- les fonctions de construction des objets composés à partir des objets de base.

Nous verrons également que pour les atomes, il faut définir un attribut "valence".

3.2.1. Les objets de base.

Seuls les mots terminaux : atome, liaison, entier ont une importance sémantique. Les autres terminaux (po, pf, deux, point) n'ont qu'une valeur syntaxique.

L'analyseur lexicographique fournit un code entier, appartenant à [*1, maxatom*] pour les atomes. Pour distinguer les deux formes des atomes, atome d'une lettre et atome entre apostrophes, nous avons choisi de

- coder de 1 à 26 les premiers,
- de stocker, au fur et à mesure de la rencontre d'un nouvel atome entre apostrophes, sa chaîne de caractères alphanumériques (sans les apostrophes) dans un tableau *ensatom* [*27..maxatom*] dont les éléments sont d'un type *T* qu'il reste à définir.

Ceci implique que *maxatom* doit être supérieur à 27. Ces atomes sont codés par leur indice dans le tableau *ensatom*.

Nous définissons alors l'objet *atome* par :

type atome = 1..maxatom ;

Nous avons seulement trois types de liaison, simple, double ou triple reconnu par l'analyseur lexicographique. L'objet *liaison* est donc défini par :

type liaison = (simple, double, triple) ;

La valeur d'un mot terminal entier est calculée au niveau lexicographique. Nous associons donc le *type entier* au terminal entier.

3.2.2. Les objets composés.

Il faut associer un objet aux non terminaux <molécule>, <groupe>, <molsym>, <radicallibre>, <listesubst>, <subst>, (<L> définit une *liaison* et <M> un *entier* ; nous n'associons pas d'objet à l'axiome <constituant> car il est inutile de définir un objet unique pour les molécules et les radicaux libres).

Tous ces objets sont représentés par des graphes, ou des ensembles de graphes dans le cas de <listesubst>, non orientés. Mais la notation linéaire que nous avons décrite exige le choix d'un atome foyer. La proposition 1 (cf. § 1) exprime alors qu'il existe un sous graphe partiel unique du graphe qui soit une arborescence dont la racine est le noeud associé à l'atome foyer. Grâce à cette proposition, et parce que nous avons éliminé les constituants cycliques, la représentation interne des constituants peut être de type arborescence.

Nous appelons :

- *arbre* l'objet associé à <groupe> ;
- *forêt* l'objet associé à <listesubst> ;
- *arbrelié* l'objet associé à <subst>.

Une décomposition naturelle de ces objets, calquée sur les règles de production de la grammaire, est la suivante :

type arbre = (at : atome ; f : forêt) ;
arbrelié = (l : liaison ; ar : arbre) ;
forêt = collection de arbrelié ;

Nous introduisons le type structure *collection de* au lieu des types usuels *ensemble*, *liste*, *tableau*, ... pour les raisons suivantes :

- une *forêt* contient a priori un nombre quelconque d'*arbreliés* ;
- plusieurs *arbreliés* peuvent être identiques dans une même *forêt* ;
- les *arbreliés* ne sont pas ordonnés.

Nous définissons maintenant les objets associés aux molécules et aux radicaux libres.

- Une molécule sous forme non symétrique, produite par le non terminal <groupe>, est construite avec un objet de type *arbre* ; une molécule

sous forme symétrique est définie par sa *moitié*, qui inclut la *liaison* centre de symétrie ; cette moitié est de type *arbrelié*.
Nous avons donc :

```
type molécule = (cas symque : booléen de
                    vrai : (moitié : arbrelié) ;
                    faux : (ar : arbre)) ;
```

- Un radical libre est construit avec un objet de type *arbre*, mais son atome racine porte un électron célibataire symbolisé par un point. Cette distinction est faite en introduisant un nouveau type :

```
type radical = (ar : arbre) ;
```

3.2.3. Fonctions de construction des objets composés.

Ces fonctions de construction servent essentiellement au calcul des attributs sémantiques des mots de la grammaire. Nous donnons seulement leurs spécifications.

- Un *arbre* est construit à partir d'un atome racine *at*, et des sous-arbres regroupés dans une *forêt*. Nous définissons donc la

```
fonction enraciner (at : atome ; f : forêt) : arbre ;
```

- Un *radical* est construit à partir d'un *arbre* par la

```
fonction pointer (ar : arbre) : radical ;
```

- Un *arbrelié* est formé d'une *liaison* et d'un *arbre* ; il est construit par la

```
fonction lier (l : liaison ; ar : arbre) : arbrelié ;
```

- Une *forêt* est construite progressivement en ajoutant des objets de type *arbrelié*. On définit d'une part une variable

```
var fvide : forêt ;
```

qui ne contient aucun élément, et d'autre part la

```
fonction ajouter (n : entier ; al : arbrelié ; f : forêt) : forêt ;
```

qui ajoute à la *forêt f n arbreliés* identiques à *al*.

- La construction d'une *molécule* est faite par l'une des fonctions :

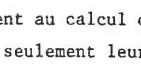
```
fonction constmolnonsym (ar : arbre) : molécule ;
fonction constmolsym (al : arbrelié) : molécule ;
```


3.2.4. Valence d'un atome.

Comme nous l'avons dit (chapitre II, § 1), un même élément chimique a toujours la même valence (pendant une même expérience). Ceci permet de trouver des erreurs, mais élimine des cas d'exception réels. Puisque la notation chimique décrit toutes les liaisons entre les atomes, rappelons que nous pouvons calculer la valence d'un atome a dans un constituant c en faisant la somme de toutes les liaisons, comptées avec leur ordre de multiplicité, de a dans c, et en ajoutant un à cette valeur si a porte un électron célibataire.

Exemples :

La valence de l'atome C dans est 4 car C a 2 liaisons simples et une liaison double.

La valence de l'atome N dans  est 3 car N a deux liaisons simples et porte un électron célibataire.

Un atome libre, comme l'hélium He a une valence nulle.

Ceci permet de faire une validation sémantique pour un constituant, en vérifiant que tous les atomes ayant le même symbole chimique ont la même valence. A priori la valeur de référence n'est pas connue, et ne peut l'être du système car le format d'un atome est libre. Deux solutions se présentent :

(1) Demander à l'utilisateur d'indiquer la valence des atomes utilisés par une déclaration du type :

```
valence 'atome' ='entier' ;
```

(2) Prendre comme valeur de référence, la première valence calculée pour un atome d'un élément chimique donné.

Nous avons choisi la deuxième solution qui est moins lourde, au risque d'avoir une valeur de référence erronée. Ceci n'est pas gênant car toute incohérence dans une notation entraîne l'arrêt de la construction du constituant analysé.

La valence d'un atome est un entier compris entre zéro et une valeur maximum donnée par une constante *maxval*. Nous réservons la valeur -1 lorsque la valence d'un atome est indéterminée. Ces valeurs sont stockées au fur et à mesure de leur calcul dans un tableau :

```
var  valatom : tableau [atome] de -1..maxval ;
```

initialisé à -1.

Le stockage est fait par une fonction sémantique

stockerval (in at : atome ; in v : entier) ;

qui doit en outre valider la valeur *v*.

L'action de *stockerval* est donc :

```
début
  si (valatom [at] = -1) et (v ⩽ maxval)
    alors % v est la valence de at %
      valatom [at] := v ;
    sinon
      si (v > maxval) ou (v ≠ valatom [at])
        alors
          'signaler l'erreur : valence erronée' ;
      fsi ;
  fsi ;
fin ; % stockerval %
```

La valence d'un atome est connue à partir de son contexte syntaxique. Son calcul est fait au moyen d'attributs. Nous aurons besoin de la

fonction rang (l : liaison) : entier ;

qui retourne le rang, c'est-à-dire l'ordre de multiplicité, de la liaison *l*.

3.2.5. Définition et calcul des attributs.

Les attributs utilisés sont définis dans le tableau suivant :

nom	nature	type	signification
at.atome	synthétisé	*atome*	code de l'atome lu par l'analyseur lexicographique
l.liaison *l.<L>*	synthétisé	*liaison*	nature de la liaison lue par l'analyseur lexicographique ou produite par <L>
n.entier *n.<M>*	synthétisé	*entier*	valeur d'un entier calculée par l'analyseur lexicographique ou produite par <M>
m.<molécule>	synthétisé	*molécule*	*molécule* produite par <molécule>
r.<radicallibre>	synthétisé	*radical*	*radical* libre produit par <radicallibre>
ar.<groupe>	synthétisé	*arbre*	*arbre* produit par <groupe>
al.<molsym> *al.<subst>*	synthétisé	*arbrelié*	*arbrelié* produit par <subst> ou définissant la *moitié* de <molsym>
f.<listesubst>	synthétisé	*forêt*	collection de tous les substituants lus par <listesubst>
v.<subst> *v.<listesubst>*	synthétisé	*entier*	contribution du ou des substituants à la valence de l'atome substitué
v. <groupe>	hérité	*entier*	valeur initiale de la valence de l'atome racine de l'*arbre* produit par <groupe> obtenue à partir du texte qui précède l'atome

Nous réécrivons les règles de production de la grammaire en insérant le calcul des attributs.

<constituant> ::= <molécule> | <radicallibre>

<molécule> ::= <groupe>
v.<groupe> := 0 ;
m.<molécule> := constmolnonsym (ar.<groupe>) ;

<molécule> ::= <molsym>
m.<molécule> := constmolsym (al.<molsym>) ;

<radicallibre> ::= point <groupe>
v.<groupe> := 1 ;
r.<radicallibre> := pointer (ar.<groupe>) ;

<molsym> ::= <L> atome deux
stockerval (at.atome, rang (l.<L>)) ;
al.<molsym> := lier (l.<L>, enraciner (at.atome, fvide)) ;

<molsym> ::= <L> po <groupe> pf deux
al.<molsym> := lier (l.<L>, ar.<groupe>) ;

<groupe> ::= atome <listesubst>
stockerval (at.atome, v.<groupe> + v.<listesubst>) ;
ar.<groupe> := enraciner (at.atome, f.<listesubst>) ;

$<listesubst>_1$ ::= <subst> <M> $<listesubst>_2$
$v.<listesubst>_1 := v.<listesubst>_2 + (n.<M> * v.<listesubst>_2)$;
$f.<listesubst>_1 := ajouter (n.<M>, al.<subst>, f.<listesubst>_2)$;

<listesubst> ::= liaison <groupe>
v.<groupe> := rang (l.liaison) ;
v.<listesubst> := rang (l.liaison) ;
f.<listesubst> := ajouter (1,
lier (l.liaison, ar.<groupe>),
fvide) ;

<listesubst> ::= Λ
v.<listesubst> := 0 ;
f.<listesubst> := fvide ;

<subst> ::= atome
stockerval (at.atome, 1) ;
v.<subst> := 1 ;
al.<subst> := lier (simple, enraciner (at.atome), fvide) ;

<subst> ::= po <L> <groupe> pf
v.<groupe> := rang (l.<L>) ;
v.<subst> := rang (l.<L>) ;
al.<subst> := lier (l.<L>, ar.<groupe>) ;

<L> ::= liaison
l.<L> := l.liaison ;

<L> ::= Λ
l.<L> := simple ;

<M> ::= entier
n.<M> := n.entier ;

<M> ::= Λ
n.<M> := 1 ;

Remarque :

Suite au commentaire 6 du paragraphe 3.1.4., l'*arbrelié* construit à partir de la règle <listesubst> ::= liaison <groupe> est considéré comme substituant, au même titre que les *arbreliés* produits par les règles :

<subst> ::= atome | po <L> <groupe> pf

En conséquence, cet *arbrelié* appartient à la *forêt* construite par <listesubst> et qui décrit tous les substituants d'un même atome. Cela entraîne que la représentation interne d'un constituant oublie le choix des chemins principal et secondaires mis en évidence dans la notation linéaire externe. Cette perte d'information n'est pas gênante car ce choix est arbitraire.

Nous avons défini la sémantique de la grammaire aux moyens d'attributs. Nous verrons au chapitre V comment nous l'avons implantée pour lire les réactifs d'un mécanisme réactionnel.

4. REPRESENTATION INTERNE CANONIQUE.

La notation des constituants que nous avons définie n'est pas canonique, c'est-à-dire qu'à un constituant donné peut correspondre plusieurs expressions. Cette non-canonicité dépend essentiellement du choix de l'atome foyer (dans le cas des molécules), des chemins principal et secondaires, et de l'ordre d'écriture des substituants. Il s'ensuit que la représentation interne d'un constituant n'est pas unique. Cet état de fait est volontaire, car nous voulions limiter le nombre des règles d'écriture nécessaires au codage d'un constituant, et conserver une certaine souplesse vis-à-vis d'un utilisateur. Mais au niveau de la qualité de l'application et de la simplicité des algorithmes, il est utile, sinon nécessaire, de disposer d'une représentation interne canonique.

Dans ce paragraphe, nous traitons du problème de représentation interne canonique, en définissant un foyer de façon unique, et un ordre total sur les substituants.

4.1. Foyer.

Nous allons définir le foyer en établissant deux critères, et en montrant l'intérêt de choisir un foyer répondant à ces critères.

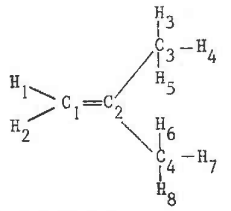
4.1.1. Définition du premier critère.

4.1.1.1. Intérêt des automorphismes dans l'arbre d'un constituant.

Nous allons montrer sur un exemple que le choix du foyer dans une molécule peut être déterminant pour la simplification de l'écriture des processus élémentaires.

Exemple :

Considérons la molécule d'isobutène de formule développée



Nous avons numéroté les atomes pour les distinguer. Les amorçages unimoléculaires appliqués à l'isobutène produisent les radicaux :

par rupture d'une des liaisons C_1-H_1, C_1-H_2,

par rupture d'une des liaisons C_2-C_3, C_2-C_4,

par rupture d'une des liaisons C_3-H_3, C_3-H_4, C_3-H_5, C_4-H_6, C_4-H_7, C_4-H_8.

Cet exemple montre qu'il existe des relations entre l'équivalence des atomes d'un constituant et l'identité de deux processus élémentaires. De telles relations sont intéressantes à exploiter, car elles peuvent permettre de connaître par avance quels sont les processus identiques.

La définition des atomes équivalents dans un constituant est faite au moyen des automorphismes de graphe.

Définition 3 :

Soient $G_1 = (X_1, \Gamma_1)$ et $G_2 = (X_2, \Gamma_2)$ deux graphes ; un isomorphisme de G_1 sur G_2, est une bijection ϕ de X_1 dans X_2 telle que :

$$\forall (x, y) \in X_1^2 , \quad x \,\Gamma_1\, y \iff \phi(x)\,\Gamma_2\,\phi(y)$$

Un automorphisme de $G = (X, \Gamma)$ est un isomorphisme de G sur lui-même.

Si les graphes G_1, G_2 et G sont étiquetés, l'application ϕ doit en outre conserver leurs étiquettes.

Définition 4 :

Soit $G = (X, \Gamma)$ un graphe. Sur X on définit une relation d'équivalence $\sim_\Gamma$ par :

$$\forall (x, y) \in X^2 , \quad x \sim_\Gamma y \iff \exists \phi \text{ automorphisme de G tel que } \phi(x) = y$$

Il est facile de vérifier que $\sim_\Gamma$ est une relation d'équivalence : notons $X/\sim_\Gamma$ l'ensemble quotient, et $\dot{x}_\Gamma$ la classe d'équivalence de x.

Lorsque G est un graphe étiqueté associé à un constituant chimique, nous parlerons d'atomes équivalents. Deux atomes équivalents ont bien sûr la même étiquette. En conséquence, tout isomorphisme entre deux radicaux libres met en correspondance leurs atomes pointés. Donc si $G = (X, \Gamma)$ est un arbre associé à un radical libre, et si x correspond à l'atome pointé, nous avons toujours :

$$\dot{x}_\Gamma = \{x\}$$

En utilisant la notation des motifs, les relations qui existent entre l'équivalence d'atomes dans un constituant et l'identité des processus s'exprime comme suit :

(Rau) - Soient A—B et A'—B' deux motifs d'une molécule ; alors les deux amorçages unimoléculaires obtenus respectivement par rupture des liaisons A—B et A'—B' sont identiques si, et seulement si $A \sim A'$ et $B \sim B'$ (ou $A \sim B'$ et $B \sim A'$).

(Rde) - Soient $\cdot A{=}B{-}C$ et $\cdot A{=}B'{-}C'$ deux motifs d'un radical libre ; alors les deux décompositions du radical obtenues respectivement par rupture des liaisons B—C et B'—C' sont identiques si, et seulement si $B \sim B'$ et $C \sim C'$.

(Rad1) - Soient $A{\equiv}B$ et $A'{\equiv}B'$ deux motifs d'une molécule et $\cdot C$ le motif d'un radical libre ; alors les deux processus d'addition obtenus en formant respectivement les liaisons B—C et B'—C sont identiques si et seulement si $A \sim A'$ et $B \sim B'$ (ou $A \sim B'$ et $B \sim A'$).

(Rad2) - Soient $A{\equiv}B$ le motif d'une molécule et $\cdot C$ le motif d'un radical libre ; alors les deux processus d'addition obtenus en formant respectivement les liaisons A—C et B—C sont identiques si, et seulement si $A \sim B$.

Remarques :

1) Nous avons noté par convention $\equiv$ une liaison simple ou double, $\equiv$ une liaison double ou triple.

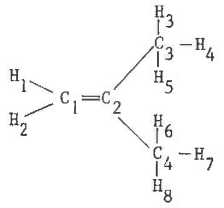
2) Nous n'avons pas de relation pour les combinaisons, car les radicaux libres ont un seul électron célibataire et donc il existe une façon unique de former une liaison entre eux. En conséquence la combinaison de deux radicaux libres est unique et produit toujours une seule molécule.

3) Suite au paragraphe 3.4.1. du chapitre II il est inutile de considérer les processus composés.

Ces relations montrent que si l'on connaît les classes d'équivalence des atomes dans un constituant, et que si l'on applique un modèle de processus aux motifs correspondants du constituant dont les atomes ne sont pas équivalents deux à deux, on obtient tous les processus différents de ce modèle à partir du constituant.

Exemple :

Reprenons l'exemple de l'isobutène.



Les motifs de la forme A—B correspondant à un amorçage unimoléculaire sont :

$$C_1-H_1,\ C_1-H_2,$$
$$C_2-C_3,\ C_2-C_4,$$
$$C_3-H_3,\ C_3-H_4,\ C_3-H_5,$$
$$C_4-H_6,\ C_4-H_7,\ C_4-H_8.$$

Les atomes de l'isobutène sont partitionnés en 5 classes d'équivalences, déterminées à partir d'un inventaire des automorphismes de l'arbre associé :

$$\{C_1\},\ \{H_1, H_2\},\ \{C_2\},\ \{C_3, C_4\},\ \{H_3, H_4, H_5, H_6, H_7, H_8\}.$$

On appliquera le modèle d'amorçage unimoléculaire à

- l'un des motifs C_1-H_1, C_1-H_2 ;
- l'un des motifs C_2-C_3, C_2-C_4 ;
- l'un des motifs C_3-H_3, C_3-H_4, C_3-H_5, C_4-H_6, C_4-H_7, C_4-H_8.

On est sûr alors d'avoir tous les amorçages unimoléculaires obtenus à partir de l'isobutène et de ne les avoir qu'une seule fois.

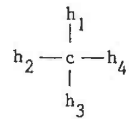
Ce résultat est intéressant car il évite de créer plusieurs fois le même processus et de comparer tout nouveau processus aux processus déjà créés, ce qui est un algorithme long et coûteux.

4.1.1.2. Automorphismes dans les arbres et les arborescences.

Les automorphismes de l'arbre d'un constituant sont intéressants pour la création des processus élémentaires, mais la représentation interne d'un constituant est celle d'une arborescence $G_r = (X, \Gamma_r)$, qui dépend du choix d'un atome foyer fait par l'utilisateur, et non celle de l'arbre $G = (X, \Gamma)$. En général les relations d'équivalence $\sim_\Gamma$ et $\sim_{\Gamma_r}$ et, par suite, les ensembles quotients $X/\sim_\Gamma$ et $X/\sim_{\Gamma_r}$ sont différents.

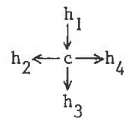
Exemple :

Considérons l'arbre $G = (X, \Gamma)$ ci-dessous, associé à la molécule de méthane CH_4 :



$$X/\sim_\Gamma = \{\{c\}, \{h_1, h_2, h_3, h_4\}\}.$$

L'arborescence $G_{h_1} = (X, \Gamma_{h_1})$ de racine h_1 est la suivante :



et nous avons :

$$X/\sim_{\Gamma_{h_1}} = \{\{c\}, \{h_1\}, \{h_2, h_3, h_4\}\}.$$

Cependant, nous allons montrer que l'on peut choisir dans tout arbre $G = (X, \Gamma)$ un noeud r tel qu'il soit possible de déterminer l'ensemble quotient $X/\sim_{\Gamma}$ à partir de $X/\sim_{\Gamma_r}$.

Proposition 3 :

Soient $G = (X, \Gamma)$ un arbre et r un point de X.

Alors :

$$\dot{r}_{\Gamma} = \{r\} \iff \sim_{\Gamma} = \sim_{\Gamma_r}$$

Nous démontrons cette proposition à partir des deux lemmes suivants.

Lemme 1 :

Avec les mêmes hypothèses nous avons :

ϕ automorphisme de $G = (X, \Gamma)$

$\iff$ ϕ isomorphisme de $G_r = (X, \Gamma_r)$ dans $G_{\phi(r)} = (X, \Gamma_{\phi(r)})$

Démonstration :

a) Condition suffisante :

Démonstration évidente avec la proposition 2.

b) Condition nécessaire :

Soit ϕ un automorphisme de $G = (X, \Gamma)$. D'après la proposition 2, nous avons :

$$\Gamma = \Gamma_r \cup \Gamma_r^{-1} = \Gamma_{\phi(r)} \cup \Gamma_{\phi(r)}^{-1}$$

d'où :

$\forall (x, y) \in X^2$, $x \Gamma_r y$ ou $y \Gamma_r x \iff \phi(x) \Gamma_{\phi(r)} \phi(y)$ ou $\phi(y) \Gamma_{\phi(r)} \phi(y)$

Si $x \Gamma_r y$ alors, nous avons successivement :

- l'unique chemin (élémentaire) de r à y dans G_r contient l'arc $x \Gamma_r y$;
- la chaîne élémentaire de r à y dans G contient l'arête $x \Gamma y$, car $\Gamma_r \subset \Gamma$;
- la chaîne élémentaire de $\phi(r)$ à $\phi(y)$ dans G contient l'arête $\phi(x) \Gamma \phi(y)$, car ϕ est un automorphisme de G.

Par définition de $G_{\phi(r)}$, la suite des points de cette chaîne est un chemin de $\phi(r)$ à $\phi(y)$ dans $G_{\phi(r)}$; donc $\phi(x) \Gamma_{\phi(r)} \phi(y)$.

Comme d'autre part $G_{\phi(r)}$ est une arborescence, nous ne pouvons pas avoir à la fois $\phi(x) \Gamma_{\phi(r)} \phi(y)$ et $\phi(y) \Gamma_{\phi(r)} \phi(x)$.

Donc :

$$\forall (x, y) \in X^2, \; x \Gamma_r y \iff \phi(x) \Gamma_{\phi(r)} \phi(y)$$

ce qui montre que ϕ est un isomorphisme de $G_r = (X, \Gamma_r)$ sur $G_{\phi(r)} = (X, \Gamma_{\phi(r)})$. □

Lemme 2 :

Avec les mêmes hypothèses nous avons :

ϕ automorphisme de $G = (X, \Gamma)$ et $\phi(r) = r$

$\iff$ ϕ automorphisme de $G_r = (X, \Gamma_r)$.

Démonstration :

Le lemme 2 est une conséquence directe du lemme 1 : il suffit simplement de remarquer qu'un automorphisme sur une arborescence laisse la racine invariante. □

Démonstration de la proposition 3 :

Si $\dot{r}_{\Gamma} = \{r\}$, tous les automorphismes de $G = (X, \Gamma)$ conservent le point r. D'après le lemme 2, tous les automorphismes de $G = (X, \Gamma)$ sont alors des automorphismes de $G_r = (X, \Gamma_r)$ et inversement, ce qui par définition est équivalent à l'identité des relations $\sim_{\Gamma}$ et $\sim_{\Gamma_r}$. □

D'après la proposition 3, si un constituant chimique contient un atome qui est l'unique représentant de sa classe d'équivalence, alors l'arborescence, obtenue en choisissant cet atome comme foyer, a le même ensemble quotient que l'arbre du constituant. Ce résultat est intéressant car pour la recherche de l'ensemble quotient, la représentation interne du constituant par cette arborescence suffit, et qu'il est plus rapide de trouver les noeuds équivalents dans une arborescence que dans un arbre.

Dans un radical libre, l'atome pointé est le seul représentant de sa classe d'équivalence, puisqu'il est le seul à porter une étiquette avec un point. Mais dans une molécule, aucun atome n'est distingué par son étiquette, en général.

Le problème posé est donc :

"Dans une molécule quelconque, existe-t-il un atome qui est le seul représentant de sa classe d'équivalence ?"

La réponse est non : par exemple dans la molécule d'hydrogène H—H, les deux atomes sont équivalents.

En fait nous allons montrer que dans toute molécule sans cycle, il existe un atome A tel que tout atome A' équivalent à A est identique à A ou lié à A ; dans ce dernier cas la classe d'équivalence de A ne contient que deux atomes A et A', et la liaison entre A et A' est liaison de symétrie. La proposition suivante énonce ce résultat en termes de graphe.

Proposition 4 :

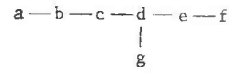
Dans tout arbre $G = (X, \Gamma)$, il existe un point a de X tel que pour tout automorphisme ϕ de G on ait $\phi(a) = a$ ou $\phi(a)\,\Gamma\,a$.

La démonstration de ce résultat et la recherche d'un tel noeud fait l'objet du paragraphe 4.1.2.5. Mais auparavant nous donnons quelques arguments en faveur du choix d'un tel noeud comme foyer.

a) Si $G = (X, \Gamma)$ est un arbre tel que pour tout automorphisme ϕ et pour tout noeud x de X on a $\exists\phi(x)\,\Gamma\,x$, alors d'après la proposition 4 il existe un noeud a de X tel que pour tout automorphisme de $G = (X, \Gamma)$ on a $\phi(a) = a$, c'est-à-dire que $\dot{a}_\Gamma = \{a\}$. D'après la proposition 3 nous avons alors :

$$\sim_\Gamma = \sim_{\Gamma_r}$$

donc les ensembles $X/\sim_\Gamma$ et $X/\sim_{\Gamma_r}$ sont identiques. a n'est pas, en général, le seul noeud de X ayant cette propriété. Par exemple dans l'arbre ci-dessous deux noeuds distincts ne sont jamais équivalents.



b) Si $G = (X, \Gamma)$ est un arbre tel qu'il existe un automorphisme ψ_o et un noeud a de X tel que $\psi_o(a)\,\Gamma\,a$, on déduit directement de la proposition 4 que $\dot{a}_\Gamma = \{a, \psi_o(a)\}$. De plus nous avons la

Proposition 5 :

Soit $G = (X, \Gamma)$ un arbre. Si il existe un automorphisme ψ_o et un noeud a de X tel que $\psi_o(a)\,\Gamma\,a$, alors :

(1) $\psi_o^2(a) = a$

(2) a et $\psi_o(a)$ sont les seuls noeuds de X vérifiant $x\,\Gamma\,\psi_o(x)$.

(3) $\psi_o(a)\,\Gamma\,a$ est une arête de symétrie, c'est-à-dire que les deux composantes connexes, $G_1 = (X_1, \Gamma_1)$ et $G_2 = (X_2, \Gamma_2)$, obtenues à partir de G en supprimant l'arête $a\,\Gamma\,\psi_o(a)$, sont isomorphes, et $\psi_o(X_1) = X_2$, $\psi_o(X_2) = X_1$.

(4) si on note Ψ l'ensemble des automorphismes ψ de G tels que $\psi(a) = \psi_o(a)$ et Φ l'ensemble des automorphismes ϕ de G tel que $\phi(a) = a$ alors :

$$\forall\psi \in \Psi,\ \exists\phi \in \Phi \text{ tel que } \psi = \psi_o \circ \phi$$
$$\forall\phi \in \Phi,\ \phi_{|X_1} \text{ est un automorphisme de } G_1.$$

Démonstration :

(1) Soit $[\psi_o^2(a) = x_o, x_1, \ldots, x_n = a]$, $n \geqslant 0$, une chaîne de G reliant $\psi_o^2(a)$ et a. Comme ψ_o est un automorphisme de G et $a\,\Gamma\,\psi_o(a)$, nous avons $\psi_o(a)\,\Gamma\,\psi_o^2(a)$. Dans G nous pouvons donc construire la chaîne $[a, \psi_o(a), \psi_o^2(a) = x_o, x_1, \ldots, x_n = a]$, dont les extrêmités coïncident. Nécessairement $a = \psi_o^2(a)$ car sinon cette chaîne serait un cycle.

(2) Supposons qu'il existe un noeud b de X distinct de a et $\psi_o(a)$ tel que $b\,\Gamma\,\psi_o(b)$. Nous avons alors $\psi_o^2(b) = b$ et $\psi_o(b)$ distinct de a et $\psi_o(a)$.

1er cas : $a\,\Gamma\,b$

Nous avons aussi $\psi_o(a)\,\Gamma\,\psi_o(b)$ car ψ_o est un automorphisme de G. La suite des points $[a, \psi_o(a), \psi_o(b), b, a]$ est alors un cycle de G, ce qui est absurde, puisque G est un arbre.

2ème cas : $\exists a\,\Gamma\,b$

Soit $[a, x_1, x_2, \ldots, x_n, b]$, $n \geqslant 1$, la chaîne élémentaire de a à b dans G.

$$\forall i \in [1, n],\ x_i \neq a \text{ et } x_i \neq b$$

ψ_o étant un automorphisme, la suite de points $[\psi_o(a), \psi_o(x_1), \ldots, \psi_o(x_n), \psi_o(b)]$ est une chaîne élémentaire de G qui lie $\psi_o(a)$ et $\psi_o(b)$. Comme, d'autre part, $\psi_o(a)\ \Gamma\ a$ et $\psi_o(b)\ \Gamma\ b$, nous pouvons construire une chaîne de G, $[a, x_1, \ldots, x_n, b, \psi_o(b), \psi_o(x_n), \ldots, \psi_o(x_1), \psi_o(a), a]$, dont les extrêmités sont identiques.

En outre nous avons :

$$(n = 1 \text{ et } a \neq b) \quad \text{ou} \quad (n \geq 2 \text{ et } x_2 \neq a)$$

Cette chaîne est donc un cycle, ce qui est absurde puisque G est un arbre.

(3) La suppression de l'arête $a\ \Gamma\ \psi_o(a)$ correspond à définir sur X la relation $\Gamma \cap \overline{\alpha(a, \psi_o(a))}$ où $\alpha(a, \psi_o(a))$ est la relation binaire définie sur X^2 par :

$$\forall (x, y) \in X^2, \quad x\ \alpha(a, \psi_o)\ y \iff \{x, y\} = \{a, \psi_o(a)\}$$

$G_1 = (X_1, \Gamma_1)$ et $G_2 = (X_2, \Gamma_2)$ sont les deux composantes connexes de $G' = (X, \Gamma \cap \overline{\alpha(a, \psi_o(a))})$. Nous choisissons par exemple X_1 l'ensemble qui contient a et X_2 celui qui contient $\psi_o(a)$.

Pour montrer que G_1 et G_2 sont isomorphes, nous montrons que :

- ψ_o est un automorphisme de G' ;
- l'image par ψ_o de tout point de X_1 est dans X_2 ;
- l'image par ψ_o de tout point de X_2 est dans X_1.

Pour tout couple (x, y) de points de X nous avons :

$$\begin{aligned} x\ \Gamma \cap \overline{\alpha(a, \psi_o(a))}\ y &\iff x\ \Gamma y \text{ et } \{x, y\} \neq \{a, \psi_o(a)\} \\ &\iff \psi_o(x)\ \Gamma\ \psi_o(y) \text{ et } \{\psi_o(x), \psi_o(y)\} \neq \{\psi_o(a), \psi_o^2(a)\} \\ &\iff \psi_o(x)\ \Gamma\ \psi_o(y) \text{ et } \{\psi_o(x), \psi_o(y)\} \neq \{a, \psi_o(a)\} \\ &\iff \psi_o(x)\ \Gamma \cap \overline{\alpha(a, \psi_o(a))}\ \psi_o(y) \end{aligned}$$

Donc ψ_o est un automorphisme de G'.

Soit x un point quelconque de X_1. Alors il existe une chaîne $[x, \ldots, a]$ reliant x et a dans G_1, donc dans G'.

Comme ψ_o est un automorphisme de G', $[\psi_o(x), \ldots, \psi_o(a)]$ est une chaîne de G' reliant $\psi_o(x)$ et $\psi_o(a)$.

$\psi_o(x)$ et $\psi_o(a)$ appartiennent donc à la même composante connexe de G', c'est-à-dire à G_2.

D'où $\psi_o(X_1) \subset X_2$.

De la même façon on montre que $\psi_o(X_2) \subset X_1$.

Comme d'autre part $\{X_1, X_2\}$ est, par construction, une partition de X et que ψ_o est une bijection sur X,

nous en déduisons que :

$$\psi_o(X_1) = X_2 \quad \text{et} \quad \psi_o(X_2) = X_1$$

(4). En effet si ψ est un automorphisme de Ψ alors $\psi_o^{-1} \circ \psi$ est un automorphisme de G et nous avons :

$$\psi_o^{-1} \circ \psi(a) = \psi_o^{-1} \circ \psi_o(a) = a$$

donc $\psi_o^{-1} \circ \psi \in \Phi$.

. Démonstration analogue à (3), mais comme $\phi(a) = a$, l'image par ϕ de tout point de X_1 est dans X_1. □

Si $G = (X, \Gamma)$ est un arbre associé à une molécule, qui vérifie les hypothèses de la proposition 5, la liaison associée à l'arête $\{a, \psi_o(a)\}$ est arête de symétrie. La représentation interne d'une telle molécule peut être de type arbrelié, dont le champ *l* contient l'étiquette de la liaison de symétrie, et dont le champ *ar* est la représentation de l'arborescence G_{1a}. G_{1a} ne contient que la moitié des noeuds de X, mais tous les noeuds de G_2 ont un équivalent dans G_1. X_1 contient donc un représentant de chaque classe d'équivalence de $X/\sim_\Gamma$, et nous avons :

$$\forall x \in G,\ \dot{x}_\Gamma = \underline{si}\ x \in X_1\ \underline{alors}\ \dot{x}_{\Gamma_1} \cup \{\psi_o(y),\ y \in \dot{x}_{\Gamma_1}\}$$
$$\underline{sinon}\ \dot{\psi_o(x)}_{\Gamma_1} \cup \{\psi_o(y),\ y \in \dot{\psi_o(x)}_{\Gamma_1}\}$$

Dans un arbre $G = (X, \Gamma)$, nous choisirons donc comme foyer un noeud r de X, tel que :

$$(C1)\ ((\dot{r}_\Gamma = \{r\}) \quad \text{ou} \quad (\dot{r}_\Gamma = \{r, s\} \quad \text{et} \quad r\ \Gamma s))$$

Nous appellerons foyer simple un noeud r tel que $(\dot{r}_\Gamma = \{r\})$ et foyer double un noeud r tel que $(\dot{r}_\Gamma = \{r, s\}$ et $r\ \Gamma s)$. Nous pourrons alors déterminer les classes d'équivalence des atomes pour Γ à partir de celles définies pour Γ_r. Cependant si l'existence d'un foyer double dans G implique son unicité, plusieurs noeuds d'un arbre peuvent répondre au critère de foyer simple. Nous allons donc définir un critère plus strict.

4.1.2. Définition du second critère.

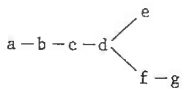
4.1.2.1. Minimisation du parenthésage de la notation des constituants : deuxième critère pour le choix du foyer.

Nous avons déjà déterminé un critère pour le choix du foyer orienté vers la simplification des algorithmes de création des processus élémentaires. Mais dans des molécules, qui ne sont pas "très régulières", beaucoup d'atomes peuvent répondre à ce critère. Nous allons donc en définir un second, beaucoup plus subjectif puisqu'il est établi à partir de considérations sur la notation externe. Mais il a l'avantage de n'accepter qu'un ou deux atomes candidats au foyer, et d'être compatible avec le premier critère, sauf pour quelques molécules ayant des atomes ou des liaisons de symétrie.

Au lieu de raisonner sur la formule développée d'un constituant, nous raisonnerons sur l'arbre associé. La notation d'un arbre est calquée sur celle d'un constituant, mais l'étiquette d'un noeud est remplacée par son nom, et les arêtes sont notées comme des liaisons simples. Nous ne tenons pas compte des règles de factorisation des substituants et de suppression des parenthèses.

Exemple :

Soit l'arbre G = (X, E) représenté ci-dessous



Une notation de G, de foyer c et de chemin principal [c, b, a] est :

$$N = c(d(e)\ /f\ /g)\ /b\ /a\ .$$

Le choix du foyer et des chemins dans un arbre influence directement le nombre de parenthèses imbriquées nécessaires à sa notation. Le second critère que nous avons retenu pour le foyer est la minimisation de ce nombre. Accessoirement ce critère va également servir à caractériser les chemins à choisir. Ce nombre est lié directement à une fonction définie pour tous les points d'une arborescence et que nous avons appelée complexité.

4.1.2.2. Complexité des points d'une arborescence.

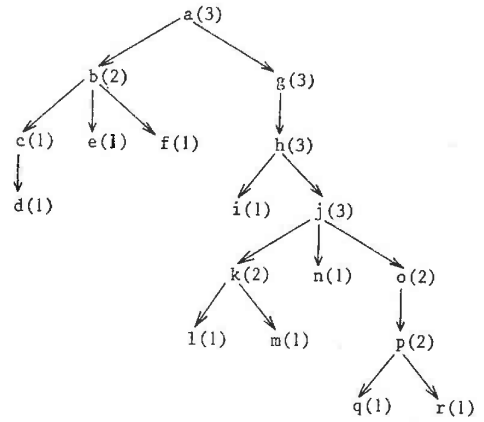
Définition 5 :

Soit A = (X, Γ) une arborescence ; on définit la complexité d'un point x de X pour la relation Γ (notée c(x, Γ)) de la façon suivante (c(x, Γ) ∈ ℕ*) :

si Γ(x) = Φ
  alors c(x, Γ) := 1 ;
  sinon m := max {c(y, Γ), y ∈ Γ(x)} ;
    si |{u ∈ Γ(x) | c(u, Γ) = m}| = 1 ;
      alors c(x, Γ) := m ;
      sinon c(x, Γ) := m + 1 ;
    fsi ;
fsi ;

Exemple :

Dans l'arborescence A = (X, Γ) ci-dessous, la complexité de chaque noeud est notée entre parenthèses.



Remarque :

La définition de c(x, Γ) ne dépend que de la sous-arborescence de racine x, notée A^x, définie par la restriction de Γ à Γ*(x) (A^x = (Γ*(x), Γ | Γ*(x))) ; on convient donc de dire que c(x, Γ) est la

complexité de cette sous-arborescence ; en particulier si r est la racine de A, $c(r, \Gamma)$ est la complexité de A.

1ère conséquence de la définition :

Soient $A = (X, \Gamma)$ une arborescence de racine r, u un point de X. Pour tout point x de la sous-arborescence A^u, nous avons :

$$c(x, \Gamma) \leqslant c(u, \Gamma)$$

2ème conséquence de la définition :

Tout point x d'une arborescence $A = (X, \Gamma)$ a au plus un successeur y tel que $c(x, \Gamma) = c(y, \Gamma)$.

Démonstration :

Si x avait deux successeurs y_1 et y_2 de complexité $c(x, \Gamma)$, la complexité de x serait au moins égale à $c(x, \Gamma) + 1$. □

Dans une arborescence $A = (X, \Gamma)$ de racine r, on peut donc joindre tous les noeuds de complexité $c(r, \Gamma)$ par un chemin. En choisissant un tel chemin comme chemin principal, la notation obtenue a un nombre minimum de parenthèses imbriquées. Les chemins secondaires sont choisis de la même façon dans les arborescences du sous-graphe engendré par les noeuds qui n'appartiennent pas au chemin principal.

Exemple :

Dans l'arborescence G de l'exemple précédent choisissons le chemin [a, g, h, j, k, l] qui contient tous les points de complexité 3. Une notation de G est :

$$N_1 = a(b(e)(f)\ /\ c\ /\ d)\ /\ g\ /\ h(i)\ /\ j(n)(o\ /\ p(q)(r))/\ k(m)\ /\ l$$

N_1 a au maximum deux couples de parenthèses imbriquées.

Nous aurions pu arrêter le chemin au noeud j, dernier noeud de complexité 3. La notation N_2 obtenue est alors :

$$N_2 = a(b(e)(f)\ /\ c\ /\ d)\ /\ g\ /\ h(i)/j(n)(o\ /\ p(q)(r))\ \ (k(m)\ /\ l)$$

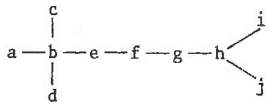
N_2 a un couple de parenthèses de plus que N_2 (autour de "k(m)/l"), mais le nombre maximum de parenthèses imbriquées est toujours 2.

On peut vérifier que le nombre de parenthèses imbriquées nécessaires à la notation d'une arborescence $A = (X, \Gamma)$ de racine a est égal à $c(r, \Gamma) - 1$. Si le chemin principal ne contient pas tous les noeuds de complexité $c(r, \Gamma)$, alors ce nombre augmente. Ceci provient du fait que dans les arborescences du sous-graphe de A, engendré par les noeuds qui n'appartiennent pas au chemin principal, il en existe une de complexité $c(r, \Gamma)$.

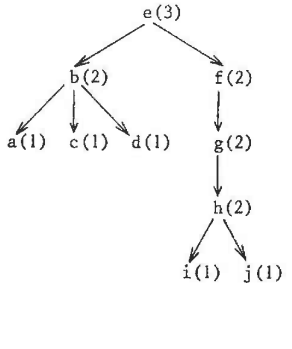
La complexité est définie pour une arborescence dont la racine est fixée. Dans un arbre $G = (X, \Gamma)$, à chaque noeud de X, nous pouvons associer une arborescence. Les complexités de deux arborescences G_s et G_r, de racines respectives r et s, déduites de G, n'ont pas toujours la même complexité.

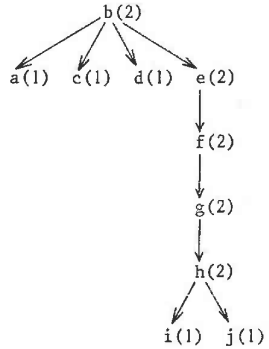
Exemple :

Soit l'arbre G suivant :



Nous avons représenté ci-dessous les arborescences G_e et G_b, de racines e et b, et indiqué la complexité de chaque noeud.





Nous avons $c(e, \Gamma_e) = 3$ et $c(b, \Gamma_b) = 2$.

Pour un graphe $G = (X, \Gamma)$, si l'on minimise les valeurs des fonctions $[u \mapsto c(., \Gamma_u),\ u \in X]$, c'est-à-dire si l'on trouve un noeud r de X tel que :

(C2) $\forall u \in X, \forall x \in X, c(x, \Gamma_r) \leqslant c(x, \Gamma_u)$,

alors la notation de G de foyer r aura un minimum de parenthèses imbriquées. Cette propriété définit notre second critère pour le choix du foyer.

Remarque :

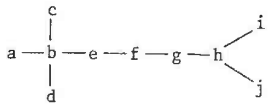
Pour minimiser le nombre de parenthèses imbriquées dans une notation, le critère suivant suffisait :

" choisir comme foyer de $G = (X, \Gamma)$ un noeud r de X tel que :
$\forall u \in X, c(u, \Gamma_u) \geqslant c(r, \Gamma_r)$ ".

Mais beaucoup de noeuds répondent à ce critère, et ces noeuds ne vérifient pas en général le premier critère.

Exemple :

Soit l'arbre $G = (X, \Gamma)$ suivant :



$\forall x \in \{a, b, c, d, h, i, j\}, \quad c(x, \Gamma_x) = 2$
$\forall x \in \{e, f, g, h\}, \quad c(x, \Gamma_x) = 3$

Donc G a 7 noeuds sur 11 qui minimisent la fonction $[x \longmapsto c(x, \Gamma_x)]$.

De plus les noeuds a, c, d d'une part et i, j d'autre part sont équivalents dans G et ne peuvent donc pas vérifier le premier critère.

Pour choisir un foyer dans un arbre $G = (X, \Gamma)$ nous chercherons d'abord un noeud répondant au premier critère. Si plusieurs noeuds sont candidats au foyer, alors nous choisirons un noeud qui répond aussi au second critère.

4.1.2.3. Majoration du nombre de points satisfaisant le second critère.

Dans ce paragraphe nous allons montrer que le nombre de noeuds r d'un arbre $G = (X, \Gamma)$ vérifiant le critère

(C2) $\forall u \in X, \forall x \in X, c(x, \Gamma_r) \leqslant c(x, \Gamma_u)$

est au plus deux. Ce résultat amène à considérer toutes les arborescences

obtenues à partir de G en prenant successivement chaque noeud de X comme racine. Nous allons d'abord exposer quelques lemmes qui décrivent les transformations des arborescences dues à un changement de racine.

Lemme 3 :

Soient $G (X, \Gamma)$ un arbre, r et s deux points distincts de G. Soit α l'unique chaîne élémentaire de r à s dans G_r ; la relation Γ_s est définie par :

$\forall (x, y) \in X^2, \; x \Gamma_s y =$ *si* $\{x, y\}$ est une arête de α
alors $\neg\, x \Gamma_r y$;
sinon $x \Gamma_r y$;
fsi ;

Démonstration :

Nous définissons sur X la relation Δ par :

$x \Delta y =$ *si* $\{x, y\}$ est une arête de α
alors $\neg\, x \Gamma_r y$;
sinon $x \Gamma_r y$;
fsi ;

Nous allons montrer que cette relation est identique à Γ_s c'est-à-dire qu'elle définit sur X une arborescence de racine s et qu'elle est incluse dans Γ.

Soit E l'ensemble des noeuds de α. D'après la définition de Δ, nous avons :

$$\forall (x, y) \in X^2, \; x \Delta y \iff (\{x, y\} \text{ arête de } \alpha \text{ et } \neg(x \Gamma_r y)) \text{ ou } (\neg(\{x, y\} \text{ arête de } \alpha) \text{ et } x \Gamma_r y)$$

$\{x, y\}$ est une arête de α si et seulement si x et y sont dans E et $x \Gamma y$. Comme $\Gamma = \Gamma_r \cup \Gamma_r^{-1}$ (proposition 2), nous avons d'une part :

$$\begin{aligned} &\{x, y\} \text{ arête de } \alpha \text{ et } \neg\, x \Gamma_r y \\ &\iff x \in E \text{ et } y \in E \text{ et } x \Gamma y \text{ et } \neg\, x \Gamma_r y \\ &\iff x \in E \text{ et } y \in E \text{ et } y \Gamma_r x, \end{aligned}$$

et d'autre part :

$$\begin{aligned} &\neg(\{x, y\} \text{ arête de } \alpha) \text{ et } x \Gamma_r y \\ &\iff (x \notin E \text{ ou } y \notin E \text{ ou } \neg\, x \Gamma y) \text{ et } x \Gamma_r y \\ &\iff (x \notin E \text{ ou } y \notin E) \text{ et } x \Gamma_r y \end{aligned}$$

Donc :

$$\forall (x, y) \in X^2,\ x \Delta y \iff (x \in E \text{ et } y \in E \text{ et } y \Gamma_r x) \text{ ou } ((x \notin E \text{ ou } y \notin E) \text{ et } x \Gamma_r y)$$

Par suite nous avons :

$$y \in E \implies \Delta^{-1}(y) = \{x \in E,\ y \Gamma_r x\} + \{x \notin E \text{ et } x \Gamma_r y\}$$
$$= (\Gamma_r(y) \cap E) + (\Gamma_r^{-1}(y) \cap (X - E))$$
$$y \notin E \implies \Delta^{-1}(y) = \Gamma_r^{-1}(y)$$

Nous allons calculer le cardinal de $\Delta^{-1}(y)$ successivement pour (a) $y = r$, (b) $y = s$, (c) $y \in E - \{r, s\}$ et (d) $y \notin E$.

(a) r est un point de E, donc :

$$\Delta^{-1}(r) = (\Gamma_r(r) \cap E) + (\Gamma_r^{-1}(r) \cap (X - E))$$

r est l'origine du chemin α, donc $\Gamma_r(r) \cap E$ contient un point unique ; comme c'est la racine de G_r, $\Gamma_r^{-1}(r)$ est vide ; d'où

$$|\Delta^{-1}(r)| = 1.$$

(b) s est un point de E, donc :

$$\Delta^{-1}(s) = (\Gamma_r(s) \cap E) + (\Gamma_r^{-1}(s) \cap (X - E))$$

Comme s est l'extrêmité du chemin α, tous ses successeurs dans G_r appartiennent à X - E, et son unique prédécesseur dans G_r appartient à E ; par suite :

$$|\Delta^{-1}(s)| = 0.$$

(c) Soit y un élément de E distinct de r et s.

$$\Delta^{-1}(y) = (\Gamma_r(y) \cap E) + (\Gamma_r^{-1}(y) \cap (X - E))$$

y a un seul successeur dans E, et son unique prédécesseur est dans E, donc :

$$|\Delta^{-1}(y)| = 1.$$

(d) Enfin, si y n'appartient pas à E, alors :

$$\Delta^{-1}(y) = \Gamma_r^{-1}(y).$$

Comme G_r est une arborescence de racine r, et que r est dans E, nous avons :

$$|\Delta^{-1}(y)| = 1.$$

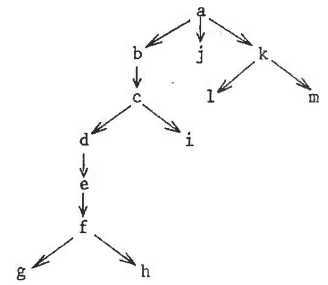
La relation Δ définit sur X une arborescence de la racine s. D'autre part son expression montre que :

$$x \Delta y \implies y \Gamma_r x \text{ ou } x \Gamma_r y$$
$$\implies x \Gamma_r \cup \Gamma_r^{-1} y$$
$$\implies x \Gamma y$$

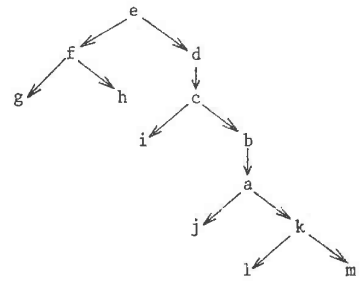
Donc Δ est incluse dans Γ. La relation Δ est donc identique à Γ_s. □

Exemple :

Soit l'arborescence $G_a = (X, \Gamma_a)$ de racine a suivante :



L'arborescence G_e de racine e est obtenue en inversant le sens de toutes les arêtes du chemin de a à e dans G_a qui sont {a, b}, {b, c}, {c, d}, {d, e}. L'arborescence G_e est donc :



Au cours de la démonstration du lemme 3 nous avons établi l'expression de $\Delta^{-1}(y)$ en fonction de $\Gamma_r(y)$ et $\Gamma_r^{-1}(y)$. De la même façon nous pouvons calculer $\Delta(y)$ en fonction de $\Gamma_r(y)$ et $\Gamma_r^{-1}(y)$. Nous donnons donc le corollaire suivant :

Corollaire :

Soient $G = (X, \Gamma)$ un arbre, r et s deux points de G, et E l'ensemble des points de la chaîne élémentaire joignant r et s dans G. Nous avons :

$$\forall x \in E,\ \Gamma_s(x) = \Gamma_r(x) - (\Gamma_r(x) \cap E) + (\Gamma_r^{-1}(x) \cap E)$$
$$\text{et } \Gamma_s^{-1}(x) = \Gamma_r^{-1}(x) - (\Gamma_r^{-1}(x) \cap E) + (\Gamma_r(x) \cap E)$$

$$\forall x \in X - E,\ \Gamma_s(x) = \Gamma_r(x)$$
$$\text{et } \Gamma_s^{-1}(x) = \Gamma_r^{-1}(x)$$

En particulier :

$$\Gamma_s(r) = \Gamma_r(r) - (\Gamma_r(r) \cap E)$$
$$\Gamma_s^{-1}(r) = \Gamma_r(r) \cap E$$
$$\Gamma_s(s) = \Gamma_r(s) + (\Gamma_r^{-1}(s) \cap E)$$

Le lemme 4 exprime l'invariance de certaines sous-arborescences de G_r, lorsqu'on choisit une nouvelle racine.

Lemme 4 :

Soient $G = (X, \Gamma)$ un arbre, a, r et s des points de G tels que a n'a pas d'occurrence dans le chemin de r à s dans G_r. Alors les sous-arborescences de racine a de G_r et de G_s sont identiques.

Démonstration :

Soit G_r^a la sous-arborescence de racine a dans G_r : aucun point de cette sous-arborescence n'appartient à E (ensemble des points de la chaîne élémentaire de r à s). En effet supposons qu'il en existe un u, on aurait :

- u est sur l'unique chemin de r à s dans G_r ;
- a est sur l'unique chemin de r à u dans G_r ;

donc a est dans E, ce qui est contraire à l'hypothèse.

De même façon, aucun point de G_s^a n'appartient à E ; ainsi d'après le lemme 3 $G_s^a = G_r^a$. □

Comme la complexité d'un point x dans une arborescence $A = (X, \Gamma)$ de racine r ne dépend que de la sous-arborescence A^x de racine x, nous avons immédiatement le lemme suivant :

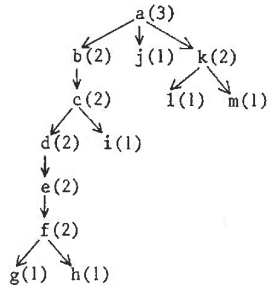
Lemme 5 :

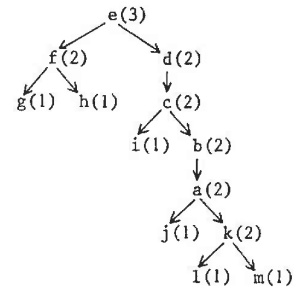
Soient $G = (X, \Gamma)$ une arborescence, a, r et s des points de G, tels que a n'a pas d'occurrence dans le chemin de r à s dans G_r ; alors :

$$c(a, \Gamma_r) = c(a, \Gamma_s).$$

Exemple :

Considérons les deux-arborescences G_a et G_e ci-dessous déduites du même arbre $G = (X, \Gamma)$





Les points {f, g, h, i, j, k, l, m} n'ont pas d'occurrence dans la chaîne élémentaire [a, b, c, d, e] de G qui joint les points a et e. Ils ont même complexité dans G_a et G_e.

Le lemme 6 ci-dessous est utilisé pour démontrer le résultat annoncé au début du paragraphe.

Lemme 6 :

Si r et s sont deux points, d'un arbre $G = (X, \Gamma)$ qui vérifient le critère (C2), alors :

(a) $\forall x \in X,\ c(x, \Gamma_r) = c(x, \Gamma_s)$

(b) $c(r, \Gamma_r) = c(s, \Gamma_r) = c(r, \Gamma_s) = c(s, \Gamma_s)$.

Démonstration :

a) r et s vérifient tous deux le critère (C2) ; nous avons donc en particulier :

$$\forall x \in X,\ c(x, \Gamma_r) \leqslant c(x, \Gamma_s)$$
$$\forall x \in X,\ c(x, \Gamma_s) \leqslant c(x, \Gamma_r)$$

d'où

$$\forall x \in X,\ c(x, \Gamma_r) = c(x, \Gamma_s)$$

b) D'après (a) :

$$c(r, \Gamma_r) = c(r, \Gamma_s)$$
$$c(s, \Gamma_r) = c(s, \Gamma_s).$$

D'après la 1ère conséquence de la définition de la complexité, nous avons :

$$c(s, \Gamma_r) \leqslant c(r, \Gamma_r)$$
$$c(r, \Gamma_s) \leqslant c(s, \Gamma_r)$$

Par suite :

$$c(r, \Gamma_r) = c(s, \Gamma_r) = c(r, \Gamma_s) = c(s, \Gamma_s)$$ □

Proposition 6 :

Soit $G = (X, \Gamma)$ un arbre ; le nombre d'éléments de X vérifiant (C2) est au plus 2.

Démonstration :

Supposons qu'il existe trois points distincts r, s et t vérifiant (C2). D'après le lemme 6, nous avons :

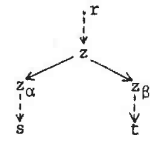
$$c(r, \Gamma_r) = c(s, \Gamma_r) = c(t, \Gamma_r)$$
$$= c(r, \Gamma_s) = c(s, \Gamma_s) = c(t, \Gamma_s)$$
$$= c(r, \Gamma_t) = c(s, \Gamma_t) = c(t, \Gamma_t)$$

On peut toujours renommer les points r, s, t, pour que, dans G_r, t n'ait pas d'occurrence dans le chemin α de r à s, et que s n'ait pas d'occurrence dans le chemin β de r à t.

(Si t a une occurrence dans α, on échange les noms de r et t ; si s a une occurrence dans β, on échange les noms de r et s).

Soient :

- γ le chemin commun à α et β (il contient au moins le point r) ;
- z le dernier noeud de γ ;
- z_α le successeur de z dans α ;
- z_β le successeur de z dans β.



Les points z_α et z_β sont distincts, puisque z est le dernier noeud du chemin commun à α et β.

D'autre part, s appartient à la sous-arborescence $G_r^{z_\alpha}$; d'après la 1ère conséquence de la définition de la complexité, nous avons :

$$c(z_\alpha, \Gamma_r) \geqslant c(s, \Gamma_r)$$

De même, t appartient à $G_r^{z_\beta}$, et :

$$c(z_\beta, \Gamma_r) \geqslant c(t, \Gamma_r).$$

Mais $c(s, \Gamma_r) = c(r, \Gamma_r)$ et $c(t, \Gamma_r) = c(r, \Gamma_r)$.

Par définition de la complexité, nous avons :

$$c(z, \Gamma_r) \geqslant c(r, \Gamma_r) + 1,$$

ce qui est absurde car r est la racine de G_r. Il ne peut donc pas exister plus de 2 points vérifiant (C2). □

Pour terminer ce paragraphe nous donnons un corollaire de cette proposition.

Corollaire :

Si r est un noeud d'un arbre $G = (X, \Gamma)$ vérifiant (C2) alors il a même complexité dans toutes les arborescences $G_u = (X, \Gamma_u)$, $u \in X$, obtenues à partir de G.

Démonstration :

Nous montrons que :

$\forall u \in X,\ c(r, \Gamma_u) = c(r, \Gamma_r)$.

Soit E l'ensemble des points de la chaîne élémentaire joignant r et u ; d'après le corollaire du lemme 3, nous avons :

$$\Gamma_u(r) = \Gamma_r(r) - (\Gamma_r(r) \cap E) + (\Gamma_r^{-1}(r) \cap E)$$
$$= \Gamma_r(r) - (\Gamma_r(r) \cap E) \quad (\text{car } \Gamma_r^{-1}(r) = \emptyset)$$

Donc $\Gamma_u(r) \subset \Gamma_r(r)$

De plus tous les points y de $\Gamma_u(r)$ n'appartiennent pas à E ; donc (lemme 5) :

$$c(y, \Gamma_r) = c(y, \Gamma_u)$$

Par suite :

$$c(r, \Gamma_u) \leqslant c(r, \Gamma_r).$$

D'autre part r vérifie (C2) donc :

$$c(r, \Gamma_r) \leqslant c(r, \Gamma_u)$$

ce qui termine la démonstration. □

Nous venons de montrer que le nombre de points vérifiant le deuxième critère est majoré par deux. Il reste à montrer l'existence de ces points.

4.1.2.4. Existence des points vérifiant le second critère.

Pour montrer l'existence des points vérifiant le second critère, nous allons d'abord donner deux propositions qui en donnent une caractérisation.

Proposition 7 :

Soit $G = (X, \Gamma)$ un arbre. Alors pour tout point r de X, nous avons :

$$r \text{ vérifie (C2)} \iff c(r, \Gamma_r) = 1$$
$$\text{ou } |\{y \in \Gamma_r(r) \mid c(y, \Gamma_r) \geqslant c(r, \Gamma_r) - 1\}| \geqslant 3$$

Démonstration :

Nous montrons successivement :

(a) $c(r, \Gamma_r) = 1 \implies$ r vérifie (C2)

(b) $c(r, \Gamma_r) \geqslant 2$ et $|\{y \in \Gamma_r(r) \mid c(y, \Gamma_r) \geqslant c(r, \Gamma_r) - 1\}| \geqslant 3$
$\implies$ r vérifie (C2)

(c) $c(r, \Gamma_r) \geqslant 2$ et $|\{y \in \Gamma_r(r) \mid c(r, \Gamma_r) \geqslant c(r, \Gamma_r) - 1\}| \leqslant 2$
$\implies$ r ne vérifie pas (C2).

a) Si r est un noeud de G tel que $c(r, \Gamma_r) = 1$, alors nous avons d'après la 1ère conséquence de la définition de la complexité :

$\forall x \in X,\ c(x, \Gamma_r) \leqslant 1.$

Ceci montre que dans ce cas r vérifie (C2).

b) Soient $G = (X, \Gamma)$ un arbre, et r un point de X tel que :

$c(r, \Gamma_r) \geqslant 2$ et $|\{y \in \Gamma_r(r) \geqslant c(r, \Gamma_r) - 1\}| \geqslant 3.$

Nous choisissons un point u quelconque de $X - \{r\}$ et nous comparons $c(x, \Gamma_u)$ et $c(x, \Gamma_r)$ pour tout x de X. Soit E l'ensemble des points du chemin de r à u dans G_r.

- Pour tout point x de $X - E$ nous avons :

$c(x, \Gamma_u) = c(x, \Gamma_r)$ (lemme 5).

- Montrons que $c(r, \Gamma_u) = c(r, \Gamma_r)$.

$\Gamma_u(r) = \Gamma_r(r) - (\Gamma_r(r) \cap E)$ (corollaire du lemme 3)

Donc $\Gamma_u(r)$ est contenu dans $\Gamma_r(r)$ et tous ses points ont même complexité dans G_r et dans G_u puisqu'ils n'appartiennent pas à E (lemme 5). Par suite :

$$c(r, \Gamma_u) \leqslant c(r, \Gamma_r).$$

D'autre part, comme $\Gamma_r(r)$ contient au moins 3 points de complexité supérieure ou égale à $c(r, \Gamma_r) - 1$, et comme $\Gamma_r(r) \cap E$ ne contient qu'un seul point, il reste dans $\Gamma_u(r)$ au moins 2 points de complexité supérieure ou égale à $c(r, \Gamma_r) - 1$. Nous avons donc :

$$c(r, \Gamma_u) \geqslant c(r, \Gamma_r)$$

et finalement : $c(r, \Gamma_u) = c(r, \Gamma_r)$.

- Enfin pour tout point y de E, nous avons, d'après la 1ère conséquence de la définition de la complexité :

$$c(y, \Gamma_r) \leqslant c(r, \Gamma_r)$$
$$c(r, \Gamma_u) \leqslant c(y, \Gamma_u)$$

Comme $c(r, \Gamma_u) = c(r, \Gamma_r)$, nous en désuisons que :

$$\forall y \in E, \quad c(y, \Gamma_r) \leqslant c(y, \Gamma_u).$$

Donc pour tous les points x de X nous avons :

$$c(x, \Gamma_r) \leqslant c(y, \Gamma_u),$$

u étant un point quelconque de $X - \{r\}$; ceci montre que r vérifie (C2).

c) Soient $G = (X, \Gamma)$ un arbre, et r un noeud de X tel que :

$$c(r, \Gamma_r) \geqslant 2 \quad \text{et} \quad |\{y \in \Gamma_r(r) \mid c(y, \Gamma_r) \geqslant c(r, \Gamma_r) - 1\}| \leqslant 2.$$

$\Gamma_r(r)$ est non vide, car $c(r, \Gamma_r) \geqslant 2$. Soit a un point de $\Gamma_r(r)$ tel que :

$$c(a, \Gamma_r) = \max \{c(y, \Gamma_r), y \in \Gamma_r(r)\}.$$

Cherchons la complexité de r dans G_a.

Les points du chemin de r à a dans G_r sont r et a.

Donc :

$$\Gamma_a(r) = \Gamma_r(r) - \{a\} \qquad \text{(corollaire du lemme 3)}$$

et $\forall x \in X - \{a, r\}, \quad c(x, \Gamma_a) = c(x, \Gamma_r)$ (lemme 5)

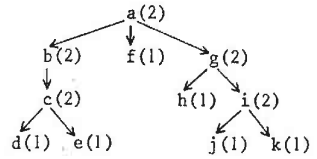
Par hypothèse, $\Gamma_r(r)$ contient au plus deux éléments de complexité $c(r, \Gamma_r) - 1$, dont un au plus peut avoir une complexité $c(r, \Gamma_r)$. $\Gamma_a(r)$ est obtenu à partir de $\Gamma_r(r)$ en enlevant à $\Gamma_r(r)$ un élément qui a la plus forte complexité. Donc $\Gamma_a(r)$ contient au plus un élément de complexité supérieure ou égale à $c(r, \Gamma_r) - 1$, et ne contient pas d'éléments de complexité $c(r, \Gamma_r)$.

Par définition de la complexité, nous avons donc :

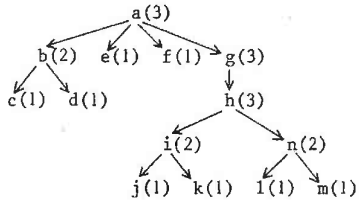
$$c(r, \Gamma_a) \leqslant c(r, \Gamma_r) - 1 \leqslant c(r, \Gamma_r).$$

Donc r ne vérifie pas (C2). □

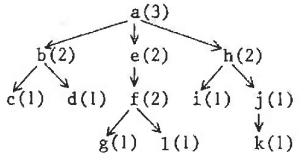
Exemple :



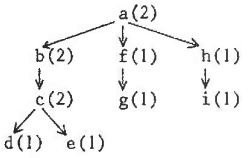
a a seulement 2 successeurs, b et g, de complexité 2 et aucun de complexité 3 : (C2) n'est pas vérifié.



a a 1 successeur, g, de complexité 3 et un seulement, b, de complexité 2 : (C2) n'est pas vérifié.



a a 3 successeurs, b, e et h, de complexité 2 : (C2) est vérifié.



a a 1 successeur, b, de complexité 2 et 2 successeurs, f et h, de complexité 1 : (C2) est vérifié.

Proposition 8 :

Soient $G = (X, \Gamma)$ un arbre, r un point de X vérifiant (C2), et α le chemin qui joint tous les points de X de complexité $c(r, \Gamma_r)$. Alors l'extrêmité de α vérifie aussi (C2) ; ce sont les seuls points vérifiant (C2).

Démonstration :

a) Si α est de longueur 0, c'est-à-dire qu'il ne contient que le point r, alors :

$$\forall x \in X - \{r\}, \quad c(x, \Gamma_r) < c(r, \Gamma_r).$$

Alors, d'après le lemme 6, aucun point distinct de r ne peut vérifier (C2).

b) α contient au moins deux points distincts et soit s son extrêmité.

Montrons que $c(s, \Gamma_s) = c(r, \Gamma_r)$

$c(s, \Gamma_s) \geqslant c(s, \Gamma_r)$ car r vérifie (C2)

$c(s, \Gamma_r) = c(r, \Gamma_r)$ par définition de s.

Donc :

$$c(s, \Gamma_s) \geqslant c(r, \Gamma_r).$$

Supposons que $c(s, \Gamma_s) > c(r, \Gamma_r)$. Ceci implique que $c(s, \Gamma_s) > 1$ et que, par conséquent, il existe un élément u de X de complexité $c(s, \Gamma_s)$ ayant deux successeurs a et b distincts dans G_s tels que :

$$c(a, \Gamma_s) \geqslant c(r, \Gamma_r)$$
$$c(b, \Gamma_s) \geqslant c(r, \Gamma_r).$$

a et b, successeurs d'un même point u, ne peuvent tous deux appartenir à un chemin de G_s, donc en particulier au chemin β de s à r dans G_s. Donc l'un des points, t, de $\{a, b\}$ n'a pas d'occurrence dans β, et nous avons

$$c(t, \Gamma_s) = c(t, \Gamma_r) \qquad \text{(lemme 5)}.$$

Or on ne peut avoir $c(t, \Gamma_r) \geqslant c(r, \Gamma_r)$, car le point t, n'ayant pas d'occurrence dans β, n'en a pas dans α.

Donc :

$$c(s, \Gamma_s) \leqslant c(r, \Gamma_r)$$

Comme d'autre part :

$$c(s, \Gamma_s) \geqslant c(r, \Gamma_r),$$

nous avons l'égalité.

- Si $c(r, \Gamma_r) = 1$, alors nous avons aussi $c(s, \Gamma_s) = 1$, ce qui, d'après la proposition 7 montre que s vérifie (C2).

- Si $c(r, \Gamma_r) \geqslant 2$, alors $\Gamma_r(s)$ contient au moins deux points distincts u et v de complexité $c(r, \Gamma_r) - 1$ dans G_r. Ces deux points n'ont pas d'occurrence dans α, donc :

$$c(u, \Gamma_s) = c(v, \Gamma_s) = c(r, \Gamma_r) - 1.$$

D'autre part :

$$\Gamma_s(s) = \Gamma_r(s) + \Gamma_r^{-1}(s) \cap E \qquad \text{(corollaire du lemme 3)}$$

où E est l'ensemble des points de α.

$\Gamma_s(s)$ contient donc au moins deux points u et v de complexité $c(r, \Gamma_r) - 1$, et le prédécesseur de s dans G_r qui est de complexité $c(r, \Gamma_r)$ puisqu'il appartient à E.

Donc si $c(r, \Gamma_r) \geqslant 2$, nous avons :

$$|\{y \in \Gamma_s(s) \mid c(y, \Gamma_s) \geqslant c(r, \Gamma_r) - 1\}| \geqslant 3$$

et comme $c(r, \Gamma_r) = c(s, \Gamma_s)$,

$$|\{y \in \Gamma_s(s) \mid c(y, \Gamma_s) \geqslant c(s, \Gamma_s) - 1\}| \geqslant 3.$$

D'après la proposition 7, ceci montre que s vérifie (C2). □

Cette proposition exprime que lorsqu'on a trouvé un noeud r de $G = (X, \Gamma)$, qui vérifie (C2), alors deux cas peuvent se présenter :

- r est le seul noeud de X de complexité $c(r, \Gamma_r)$, et alors aucun autre noeud ne vérifie (C2) ;

- il existe dans G, d'autres noeuds de complexité $c(r, \Gamma_r)$, et alors l'extrêmité du chemin qui joint tous ces noeuds vérifie également (C2). La proposition 6 indique que ce sont les deux seuls.

En fait, nous avons :

r est l'unique point de $G = (X, \Gamma)$ vérifiant (C2)

$\Leftrightarrow$ (C2a) $|\{y \in \Gamma_r(r) \mid c(y, \Gamma_r) = c(r, \Gamma_r)\}| = 0$

et $(c(r, \Gamma_r) = 1$ ou $|\{y \in \Gamma_r(r) \mid c(y, \Gamma_r) = c(r, \Gamma_r) - 1\}| \geqslant 3)$

r est un des deux points de $G = (X, \Gamma)$ vérifiant (C2)

$\Leftrightarrow$ (C2b) $|\{y \in \Gamma_r(r) \mid c(y, \Gamma_r) = c(r, \Gamma_r)\}| = 1$

et $(c(r, \Gamma_r) = 1$ ou $|\{y \in \Gamma_r(r) \mid c(y, \Gamma_r) = c(r, \Gamma_r) - 1)\}| \geqslant 2)$

La proposition (C2) est équivalente à la disjonction des deux propositions (C2a) et (C2b). Un arbre ne peut contenir à la fois un point vérifiant (C2a) et un point vérifiant (C2b). Nous classerons les arbres, et par suite les molécules, en deux catégories appelées (C2a) et (C2b).

La proposition ci-dessous prouve l'existence d'un noeud vérifiant (C2).

Proposition 9 :

Pour tout arbre $G = (X, \Gamma)$, il existe au moins un noeud s de X vérifiant (C2).

Démonstration :

La démonstration que nous proposons donne un moyen de trouver un noeud s vérifiant (C2) dans un graphe $G = (X, \Gamma)$, à partir d'une de ses arborescences $G_r = (X, \Gamma_r)$, dont la racine r ne vérifie pas (C2).

D'après la proposition 7, G_r est telle que :

$$(c(r, \Gamma_r) > 1) \quad \text{et} \quad |\{y \in \Gamma_r(r) \mid c(y, \Gamma_r) \geqslant c(r, \Gamma_r) - 1\}| \leqslant 2$$

Pour simplifier les notations, nous posons :

$$p = c(r, \Gamma_r).$$

Deux cas peuvent se présenter.

1er cas : $\Gamma_r(r)$ ne contient pas de noeuds de complexité supérieure ou égale à p, et contient deux noeuds distincts a et b de complexité $p-1$.

2ème cas : $\Gamma_r(r)$ contient un noeud unique a de complexité p et au plus un noeud b de complexité $p-1$.

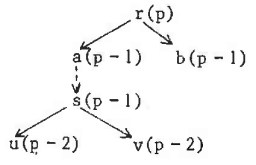
1er cas : $\Gamma_r(r)$ contient deux points a et b de complexité $p-1$.

Soient :

- α le chemin d'origine a joignant tous les points de complexité $p-1$ dans la sous-arborescence G_r^a ;
- s l'extrêmité de α ;
- E l'ensemble qui contient r et tous les points de α.

Si p est supérieur ou égal à 3, alors $\Gamma_r(s)$ contient au moins deux noeuds u et v de complexité $p-2$, et ne contient pas de noeud de complexité supérieure ou égale à $p-1$. Si p est égal à 2, alors $\Gamma_r(s)$ est vide.

Montrons que s est un point vérifiant (C2).



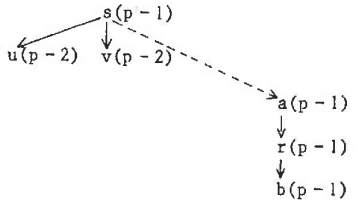
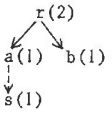


Schéma de G_r et G_s si p est supérieur ou égal à 3. (tous les noeuds non représentés de la sous-arborescence G_r^a ont une complexité inférieure ou égale à $p-2$).



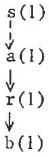


Schéma de G_r et G_s si p est égal à 2. (G_r^a ne contient que les noeuds du chemin α).

Cherchons les complexités des points de E dans G_s, qui seules peuvent changer (lemme 5).

- Complexité de r dans G_s

$\Gamma_s(r) = \Gamma_r(r) - (\Gamma_r(r) \cap E)$ (corollaire du lemme 3)

et $\Gamma_r(r) \cap E = \{a\}$

Pour obtenir $\Gamma_s(r)$, nous enlevons à $\Gamma_r(r)$ le point a. Donc $\Gamma_s(r)$ ne contient plus qu'un seul point, b, de complexité $p-1$, (tous les autres ont une complexité strictement inférieure à $p-1$) et nous avons :

$$c(r, \Gamma_s) = c(b, \Gamma_r) = p - 1.$$

- Complexité des points x de $E - \{r, s\}$ dans G_s

$$\forall x \in E - \{r, s\},\ \Gamma_s(x) = \Gamma_r(x) - (\Gamma_r(x) \cap E) + (\Gamma_r^{-1}(x) \cap E)$$

Nous remplaçons l'unique point e de $\Gamma_r(x) \cap E$ par l'unique point f de $\Gamma_r^{-1}(x) \cap E$. Les complexités de e et f dans G_r sont égales à $p-1$. Comme $c(r, \Gamma_s) = p-1$ et que tous les points de $\Gamma_s(f) \cap (X - E)$ ont une complexité inférieure ou égale à $p-2$, on peut montrer par récurrence que $c(f, \Gamma_s) = p-1$. D'autre part tous les points de $\Gamma_r(x) - (\Gamma_r(x) \cap E)$ ont une complexité inférieure ou égale à $p-2$. Donc :

$$c(f, \Gamma_s) = p - 1$$

- Complexité de s dans G_s.

$$\Gamma_s(s) = \Gamma_r(s) + (\Gamma_r^{-1}(s) \cap E).$$

Nous ajoutons à $\Gamma_r(s)$ l'unique point de $\Gamma_r^{-1}(s) \cap E$ qui a une complexité $p-1$.

Si p est égal à 2, alors ce point est l'unique point de $\Gamma_s(s)$ (puisque $\Gamma_r(s) = \emptyset$) et nous avons

$$c(s, \Gamma_s) = p - 1 = 1,$$

ce qui montre que s vérifie (C2).

Si p est supérieur ou égal à 3, alors $\Gamma_s(s)$ contient les deux points u et v de $\Gamma_r(s)$, de complexité $p-2$, et un point seulement de complexité $p-1$ (celui de $\Gamma_r^{-1}(s) \cap E$).

Ceci montre d'une part que

$$c(s, \Gamma_s) = p - 1,$$

d'autre part que s vérifie (C2).

Dans ce premier cas nous avons donc montré l'existence d'un point s de $G = (X, \Gamma)$ vérifiant (C2). L'arbre G appartient à la catégorie (C2b).

2ème cas : $\Gamma_r(r)$ contient un point unique a de complexité p.

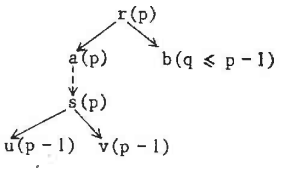
Soient :

- α le chemin de G_r qui joint tous les points de complexité p ;
- s le dernier noeud de α ;
- E l'ensemble des points de α ;
- $q = \underline{si}\ \Gamma_r(r) = \{a\}\ \underline{alors}\ 0\ \underline{sinon}\ \max \{c(x, \Gamma_r) \mid x \in \Gamma_r(r) - \{a\}\}$;
- b un point de $\Gamma_r(r)$ tel que $c(b, \Gamma_r) = q$, si $q \geq 1$.

b n'existe pas toujours, mais nous avons, par hypothèses :

$$q \leq p - 1$$
$$\forall x \in \Gamma_r(r) - \{a, b\},\ c(x, \Gamma_r) < p - 1.$$

Comme précédemment $\Gamma_r(s)$ contient deux points u et v de complexité $p-1$ dans G_r (qui existent toujours puisque $p - 1 \geq 1$), et ne contient pas de points de complexité supérieure ou égale à p.



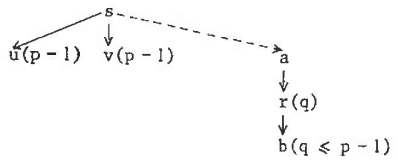


Schéma de G_r et G_s. (tous les points non représentés ont une complexité d'au plus $p-1$).

Par un raisonnement analogue au précédent, on montre que :

- $c(r, \Gamma_s) = \underline{si}\ \Gamma_s(r) = \phi\ \ \underline{alors}\ 1\ \underline{sinon}\ q$;

- $c(a, \Gamma_s) \leq p$ (avec l'égalité si $\Gamma_r(a) - (\Gamma_r(a) \cap E)$ contient deux points de complexité $p-1$)

- $c(x, \Gamma_s) \leq p$ pour tout point x appartenant à E ;

- $c(s, \Gamma_s) = p$, car $\Gamma_s(s)$ contient au moins les deux points u et v et que $c(u, \Gamma_s) = c(v, \Gamma_s) = p - 1$.

Soit t le successeur de s dans G_s appartenant à E ; si $c(t, \Gamma_s) \geq p - 1$, s vérifie (C2) puisqu'alors $\Gamma_s(s)$ a trois points, t, u et v de complexité supérieure ou égale à $c(s, \Gamma_s) - 1$, sinon on est ramené au premier cas. □

La démonstration de l'existence des points vérifiant (C2) est longue, mais nous allons calquer sur elle un algorithme qui recherche ces points (§ 4.3.). Auparavant nous étudions la compatibilité des deux critères.

4.1.2.5. Compatibilité des deux critères et existence des points vérifiant le premier critère.

Pour déterminer un foyer unique dans une molécule, dans le but d'obtenir une représentation canonique, nous venons d'établir deux critères, définis pour l'arbre $G = (X, \Gamma)$ associé à cette molécule :

(C1) : le foyer est un point r de X tel que

$$\dot{r}_\Gamma = \{r\} \quad \text{ou} \quad (\dot{r}_\Gamma = \{r, s\} \quad \text{et} \quad r \Gamma s)$$

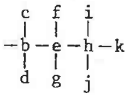
(C2) : le foyer est un point r de X tel que

$$\forall u \in X, \forall x \in X,\ c(x, \Gamma_r) \leq c(x, \Gamma_u).$$

Ces deux critères ne sont pas toujours compatibles.

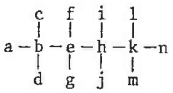
Exemple :

L'arbre $G = (X, \Gamma)$ associé à la molécule de propane $CH_3-CH_2-CH_3$ est le suivant :



On peut vérifier que e est le noeud unique vérifiant (C1) et que les noeuds b et h sont les deux noeuds vérifiant (C2).

Il en est de même pour l'arbre suivant associé à la molécule de n-butane $CH_3-CH_2-CH_2-CH_3$



Les noeuds e et h vérifient (C1) et les noeuds b et k vérifient (C2).

En fait, nous allons montrer que :

- dans un arbre $G = (X, \Gamma)$ de la catégorie (C2a), le noeud r de X vérifiant (C2) vérifie aussi (C1) ;

- dans un arbre $G = (X, \Gamma)$ de la catégorie (C2b), si les deux noeuds r et s vérifiant (C2) ne sont pas équivalents dans G, alors ils vérifient (C1), sinon le milieu de la chaîne élémentaire de G qui joint r et s (qui peut contenir un ou deux noeuds selon la parité de sa longueur) vérifie (C1) ; dans ce dernier cas nous donnerons la priorité au premier critère.

La démonstration de ce résultat, donnera les éléments nécessaires à celle de la proposition 4 (§ 4.1.1.2.) qui assure l'existence dans tout arbre $G = (X, \Gamma)$ d'un point a de X tel que pour tout automorphisme ϕ de G on ait $\phi(a) = a$ ou $\phi(a)\,\Gamma a$.

Lemme 7 :

Soit $G = (X, \Gamma)$ un arbre. Pour tout automorphisme ϕ de G, nous avons :

$$\forall r \in X, \forall x \in X, c(x, \Gamma_r) = c(\phi(x), \Gamma_{\phi(r)})$$

Démonstration :

Ce lemme est une conséquence directe de la définition de la complexité dans un arbre et du fait que pour tout automorphisme ϕ de $G = (X, \Gamma)$ et pour tout point r de G, on a :

$$\forall x \in X, \phi(G_r^x) = G_{\phi(r)}^{\phi(x)} .$$ □

Proposition 10 :

Soit $G = (X, \Gamma)$ un arbre. Pour tout automorphisme ϕ de G, si r est un point de X vérifiant (C2), alors $\phi(r)$ vérifie aussi (C2).

Démonstration :

Immédiate d'après le lemme 7.

Corollaire :

Si $G = (X, \Gamma)$ est un arbre contenant un point unique r vérifiant (C2), alors :

$\forall \phi$ automorphisme de G, $\phi(r) = r$.

Si $G = (X, \Gamma)$ est un arbre contenant deux points distincts r et s vérifiant (C2), alors :

$\forall \phi$ automorphisme de G, ($\phi(r) = r$ et $\phi(s) = s$) ou ($\phi(r) = s$ et $\phi(s) = r$)

Nous pouvons maintenant démontrer la

Proposition 4 :

Dans tout arbre $G = (X, \Gamma)$, il existe un point a de X tel que pour tout automorphisme ϕ de G $\phi(a) = a$ ou $\phi(a)\,\Gamma a$.

Démonstration :

Soient G un arbre de la catégorie (C2a), et r l'unique point de G vérifiant (C2) ; alors le corollaire de la proposition 10 montre que :

$\forall \phi$ automorphisme de G, $\phi(r) = r$.

Soient maintenant $G = (X, \Gamma)$ un arbre de la catégorie (C2b), r et s les deux points vérifiant (C2), et $\alpha = [r = x_0, x_1, \ldots, x_n = s]$, $n \geqslant 1$, la chaîne élémentaire de G joignant r et s. Alors d'après le corollaire de la proposition 10, tous les automorphismes ϕ de G sont tels que :

$$(\phi(r) = r \text{ et } \phi(s) = s) \text{ ou } (\phi(r) = s \text{ et } \phi(s) = r)$$

donc tels que :

$$\forall i \in [0, n],\ \phi(x_i) = x_i \text{ ou } \phi(x_i) = x_{n-i}$$

1er cas : La longueur n du chemin α est paire : soit $p = \frac{n}{2}$.

Alors :

$$\forall\, \phi \text{ automorphisme de G},\ \phi(x_p) = x_p.$$

2ème cas : La longueur n du chemin α est impaire : soit $p = \frac{n-1}{2}$.

Alors :

$$\phi(x_p) = x_p \text{ ou } \phi(x_p) = x_{p+1}$$

D'autre part, par définition de α, nous avons $x_p \Gamma x_{p+1}$, ce qui termine la démonstration. □

La démonstration de la proposition 10, montre non seulement l'existence des points vérifiant (C1), mais aussi la façon d'en trouver à partir des points vérifiant (C2). Nous choisirons donc le foyer d'un arbre comme suit :

(a) le foyer d'un arbre de la catégorie (C2a) est le point vérifiant (C2), donc (C1) ;

(b) le foyer d'un arbre de la catégorie (C2b) est :

(b1) l'un des points r et s vérifiant (C2), si ces points ne sont pas équivalents (ils vérifient alors aussi (C1)) ;

(b2) le milieu de la chaîne joignant les deux points r et s qui vérifient (C2) ; si la chaîne est de longueur paire, son milieu est un noeud a ; si la chaîne est de longueur impaire, son milieu est une arête $a \Gamma a'$. Si $\{r, s\}$ n'est pas une arête de G, alors le foyer vérifie (C1), mais pas (C2).

Le cas (b) nécessite de savoir si les deux noeuds r et s sont équivalents. Pour le savoir nous construisons les deux arborescences G_r et G_s, et nous les comparons avec une relation d'ordre, définie au paragraphe suivant telle que :

$$G_r = G_s \iff G_r \text{ et } G_s \text{ isomorphes}$$
$$\iff r \text{ et } s \text{ équivalents}$$

Cette relation d'ordre est aussi utilisée pour déterminer, dans le cas (b1), lequel des noeuds r et s est retenu comme foyer : nous choisirons le noeud qui donne l'arborescence la plus petite.

4.2. Relation d'ordre sur les substituants et les molécules.

La définition d'un ordre (total) sur les substituants est utile pour classer tous les substituants d'un même atome dans un constituant, et ainsi avoir une représentation interne unique d'un objet de type *forêt*. Cet ordre nous permettra également de préciser la structure de données *"collection de"*, utilisée pour définir le type *forêt*. D'autre part nous venons de voir l'utilité d'une relation d'ordre sur les molécules. Cependant seul un ordre (total) sur les molécules de forme non symétrique c'est-à-dire sur les objets de type arbre, est nécessaire.

Rappelons la définition des types *molécule* et *arbrelié*, utilisé pour les substituants, et celle des types sous-jacents :

```
type molécule = (cas symque : booléen de
                    vrai : (moitié : arbrelié) ;
                    faux : (ar : arbre)) ;
    arbrelié = (l : liaison ; ar : arbre) ;
    arbre    = (at : atome ; f : forêt) ;
    forêt    = collection de arbrelié ;
    liaison  = (simple, double, triple) ;
    atome    = 1..maxatom ;
```

Donc pour comparer deux objets de type *arbre* ou *arbrelié*, il faut savoir comparer aussi deux objets de type *forêt*, *liaison* et *atome*. La définition de ces types étant récursive, la comparaison des objets le sera aussi.

4.2.1. Ordre sur les objets de type *liaison*.

Nous définissons, de façon naturelle, l'ordre :
simple < double < triple.

4.2.2. Ordre sur les objets de type *atome*.

Les atomes sont rangés dans un tableau *ensatom*, et sont référencés, dans les constituants, par leur indice dans ce tableau. Dans un premier prototype nous avons choisi l'ordre le plus simple qu'est l'ordre induit par cet indice. Cette solution a l'inconvénient de définir un ordre qui dépend de la donnée des réactifs, et qui n'est donc pas forcément le même d'une

expérience à l'autre. Un ordre lexicographique défini par exemple sur la chaîne de caractères de l'atome serait préférable.

4.2.3. Ordre sur les objets de type *arbrelié*.

Pour comparer deux variables *al1* et *al2* de type *arbrelié*, il faut comparer leur champ *l*, et leur champ *ar*. Nous donnons la priorité à la topologie de la molécule, en comparant d'abord *ar*.

Nous avons donc :

```
∀ al1 et al2 de type arbrelié,
   al1 ⩽ al2 ≡ si  al1.ar = al2.ar
                alors  al1.l ⩽ al2.l
                sinon  al1.ar ⩽ al2.ar
               fsi ;
```

4.2.4. Ordre sur les objets de type *arbre*.

Comme pour les *arbrelié*, il faut comparer les champs *at* et *f*, en commençant par *f*. Mais le type *arbre* correspond à des arborescences sur lesquelles nous avons défini une fonction *complexité*. Le calcul de cette fonction en un point x d'une arborescence $A = (X, \Gamma)$ recherche en particulier le nombre de points u de $\Gamma(x)$ tels que

$$c(u, \Gamma) = \max \{c(y, \Gamma),\ y \in \Gamma(x)\}.$$

Donc il est intéressant de classer les arbres en fonction de leur complexité d'abord.

L'interface de la fonction *complexité* est la suivante :

```
fonction complexité (ar : arbre) : entier ;
```

La relation sur les objets de type *arbre* est alors définie comme suit :

```
∀ ar1, ar2 de type arbre,
   ar1 ⩽ ar2 ≡ si  complexité (ar1) = complexité (ar2)
                alors si  ar1.f = ar2.f
                       alors  ar1.at ⩽ ar2.at
                       sinon  ar1.f ⩽ ar2.f
                      fsi ;
               sinon  complexité (ar1) ⩽ complexité (ar2)
              fsi ;
```

4.2.5. Ordre sur les objets de type *forêt*.

Un objet de type *forêt* contient des objets de type *arbrelié*, en nombre quelconque, et certains d'entre eux peuvent être identiques. Avant d'engager une comparaison sur deux objets de type *forêt*, il est donc nécessaire de classer leurs éléments. Nous avons choisi de les classer dans l'ordre décroissant. Les premiers éléments rencontrés sont alors ceux dont la composante arbre est la plus complexe, et nous avons vu que ce sont eux qui jouent un rôle déterminant dans la recherche du foyer dans les molécules.

D'autre part, considérons une variable *ar*, de type *arbre* dont la composante *f* contient deux éléments *x* et *y* identiques de type *arbrelié*. Il est clair alors que chaque atome de *x* a un équivalent dans *y* (pour la relation Γ de l'arborescence représentée par *ar*). Nous avons donc choisi de ne faire figurer qu'une seule occurrence des éléments d'une *forêt*, et de les assortir du nombre de leurs occurrences. Le type *forêt*, initialement défini par "*collection de arbrelié*" devient donc :

type forêt = (nc : entier; descr : tableau [1..nc] de jumeaux) ;
jumeaux = (repr : arbrelié ; nb : entier) ;

Une variable *f* de type *forêt* doit toujours vérifier :

$\forall i \in [1, nc-1]$, $f.descr[i].repr > f.descr[i+1].repr$

Cette structure de données a l'avantage de ne représenter qu'un seul atome pour chaque classe d'équivalence définie dans l'arborescence représentée par une variable *ar* de type *arbre*.

Remarque :

Cette nouvelle définition de *forêt* a une influence directe sur la spécification de la

fonction ajouter (n : entier ; al : arbrelié ; f : forêt) : forêt ;

(définie au § 3.2.3.).

Nous définissons sur les objets de type *forêt* un ordre de type lexicographique.

$\forall f1, f2$ de type *forêt*,

$f1 \leqslant f2 \iff (f1.nc \leqslant f2.nc$ et $\forall i \in [1, f1.nc]$, $f1.descr[i] = f2.descr[i])$
ou $(k = min\ \{i \in \mathbb{N},\ f1.descr[i] \neq f2.descr[i]\}$ et $f1.descr[k] \leqslant f2.descr[k])$.

La relation d'ordre sur les objets de type *jumeaux* est définie par

$\forall j1, j2$ de type *jumeaux*,

j1 $\leqslant$ *j2* ≡ *si* *j1.repr = j2.repr*
alors *j1.nb* $\leqslant$ *j2.nb*
sinon *j1.repr* $\leqslant$ *j2.repr*
fsi ;

Remarques :

1) L'ordre que nous venons de définir sur les objets de type *arbrelié*, par exemple, est identique à l'ordre lexicographique défini sur les suites de 4 éléments :

complexité (al.ar), al.ar.f, al.ar.at, al.ar.l,

construites à partir des objets *al* de type *arbrelié*.

2) Soit *al* un objet de type *arbrelié* ; on peut montrer que tous les éléments *al.ar.f.descr[i].repr*, *i* de *1* à *f.nc* sont strictement inférieurs à *al*. Cet ordre est donc cohérent.

Nous avons maintenant tous les éléments nécessaires à la recherche d'une représentation canonique d'une molécule.

4.3. Algorithme pour la construction d'une représentation canonique.

L'algorithme de construction d'une représentation canonique doit chercher le foyer et classer les substituants d'un même atome. Ce classement est automatiquement réalisé par la

fonction ajouter (n : entier, al : arbrelié, f : forêt) : forêt ;

Pour les radicaux libres, le foyer est l'atome pointé et les règles de notation des constituants imposent à l'utilisateur de le choisir comme tel. Donc la représentation d'un radical libre, construite par les fonctions sémantiques de la grammaire est toujours canonique. Il en est de même des molécules écrites sous forme symétrique, puisque le premier atome rencontré correspond à son foyer double. La recherche d'un foyer ne s'impose donc que pour les molécules écrites sous forme non symétrique. L'objet construit à partir de sa lecture est alors une représentation d'une arborescence dont l'atome racine est, a priori quelconque. Le point de départ pour la recherche d'un foyer unique est une arborescence $G_u = (X, \Gamma_u)$ où u est un noeud quelconque de l'arbre $G = (X, \Gamma)$. G_u est représentée par une variable *ar* de type *arbre*. L'interface de la procédure *canonique* qui construit une représentation canonique pour une molécule est :

procédure canonique (in a : arbre ; out m : molécule) ;

L'algorithme de la procédure *canonique* est calqué sur les démonstrations de l'existence des points vérifiant les premier et second critères (§§ 4.1.2.4. et 4.1.2.5.). Il utilise les deux fonctions suivantes :

fonction compte (cp : entier, f : forêt) : entier ;

compte cherche le nombre d'éléments de *f* dont la composante arbre a une complexité égale à *cp* ; si *cp* = 0, la valeur retournée est 3.

fonction changerac (a : arbre ; cp : entier) : arbre ;

a.f contient un élément *al* tel que *complexité (al.ar) = cp* ; *changerac* construit à partir de *a* l'arborescence dont la racine est l'extrêmité du chemin qui lie tous les points de complexité *cp* dans *al.ar*.

La procédure *canonique* calcule d'abord la complexité *c* de la variable *a* de type *arbre*. En fonction des valeurs de *compte (c, a.f)*, *compte (c - 1, a.f)*, il faut distinguer les cas suivants :

(1) *compte (c, a.f) = 0 et compte (c - 1, a.f) ⩾ 3 ;*

la molécule est de catégorie (C2a) et *a* est sa représentation canonique.

(2) *compte (c, a.f) = 1 et compte (c - 1, a.f) ⩾ 2 ;*

la molécule est de catégorie (C2b) et *a.at* est un des atomes vérifiant (C2).

(3) *compte (c, a.f) = 0 et compte (c - 1, a.f) = 2 ;*

alors *changerac (a, c - 1)* vérifie (2).

(4) *compte (c, a.f) = 1 et compte (c - 1, a.f) ⩽ 1 ;*

alors *changerac (a, c)* vérifie (1), (2) ou (3).

Le traitement d'un *arbre* vérifiant (2) est fait par la procédure *chercherfoyer*. Son interface est :

```
procédure chercherfoyer (in a : arbre ; out m : molécule) ;

procédure canonique (in a : arbre ; out m : molécule) ;

var c : entier ;

début
  c := complexité (a) ;
  si compte (c, a.f) = 0
    alors
      si compte (c - 1, a.f) ⩾ 3
        alors % cas (1) ⟹ ar est la représentation canonique de m %
          m. symque := faux ; m. ar := a ;
        sinon % cas (3) ⟹ changerac (a, c - 1) vérifie (2) %
          chercherfoyer (changerac (a, c - 1), m) ;
      fsi ;
    sinon % compte (c, a.f) = 1 %
      si compte (c - 1, a.f) ⩾ 2
        alors % cas (2) %
          chercherfoyer (a, m) ;
        sinon % cas (4) %
          canonique (changerac (a, c), m) ;
      fsi ;
  fsi ;
fin ; % canonique %
```

Remarque :

La procédure *canonique* est définie de façon récursive, mais elle est appelée au plus deux fois.

La procédure *chercherfoyer* traite une molécule de la catégorie (C2b) représentée par la variable a de type *arbre* dont l'atome $a.at$ est un des points vérifiant (C2). Il faut donc construire la variable b de type *arbre*, telle que $b.at$ vérifie aussi (C2), et comparer a et b :

- si $a < b$ (respectivement $b < a$), alors a (respectivement b) est la représentation canonique de m ;

- si $a = b$, il faut chercher le milieu de la chaîne, entre les deux atomes $a.at$ et $b.at$; celui-ci est repéré par un sous-arbrelié *sal* de a et par un booléen *pair*, avec la convention :

pair = vrai ⟹ *sal.ar.at* est l'atome du milieu de la chaîne ;

pair = faux ⟹ *sal.l* est liaison de symétrie.

Nous introduisons la

```
procédure cherchermilieu (in a : arbre ;
                          out sal : arbrelié ; pair : booléen) ;

fonction chercherfoyer (in a : arbre ; out m : molécule) ;

var b    : arbre ;
    sal  : arbrelié ;
    pair : booléen ;

début
  b := changerac (a, complexité (a)) ;
  si a = b
    alors cherchermilieu (a, sal, pair) ;
      cas pair de
        vrai : début
                 m.symque := vrai ; m.moitié := sal ;
               fin ;
        faux : début
                 m.symque := faux ; m.ar := s.al.ar ;
               fin ;
      fincas ;
```



```
    sinon m.symque := faux ;
      si a < b
        alors m.ar := a ;
        sinon m.ar := b ;
      fsi ;
  fsi ;
fin ; % chercherfoyer %
```

L'élaboration de cet algorithme a nécessité un long développement. Mais il a une certaine efficacité pour plusieurs raisons :

- il ne nécessite à aucun moment la représentation d'un graphe symétrique, mais seulement celle d'une arborescence, moins gourmande en place mémoire ;

- dans le pire des cas, le nombre des nouvelles arborescences construites à partir de l'arborescence initiale est 3 ;

- la définition de la fonction *complexité* permet d'évaluer la marche à suivre pour obtenir la représentation canonique par le seul examen de la complexité de trois de ses successeurs ; cet algorithme n'est donc pas de type itératif ;

- la représentation obtenue met en évidence l'ensemble quotient des atomes d'un constituant, et seul un atome de chaque classe est représenté ; nous verrons au chapitre suivant que les algorithmes de création des processus élémentaires y gagnent en efficacité ;

- enfin les critères retenus pour le choix du foyer utilisent principalement les automorphismes dans les arbres, et donc un tel algorithme peut être utilisé pour d'autres problèmes que la représentation des molécules.

CHAPITRE IV

Création des processus élémentaires

Pour clore la description algorithmique de l'application "Construction de mécanismes réactionnels" il reste à créer les processus élémentaires. Ce chapitre est consacré à la définition des procédures *auelt*, *adelt*, *deelt*, *coelt*.

1. RAPPELS DE L'INTERFACE DES PROCEDURES *AUELT, ADELT, DEELT, COELT*.

Ces procédures ont été annoncées dans le chapitre II. Les paramètres mis à jour *lastm*, *lastr*, *lastp* désignent toujours les indices des derniers éléments des tableaux *ensmol*, *ensrad* et *ensproc*.

procédure auelt (in m : molécule ;
inout lastr : indxcnst ; inout lastp : indxpr) ;

auelt cherche tous les amorçages unimoléculaires créés à partir de la molécule *m* et les imprime.

procédure adelt (in m : molécule ; in r : radical ;
inout lastr : indxcnst ; inout lastp : indxpr) ;

adelt cherche toutes les additions créées à partir de la molécule *m* et du radical *r* et les imprime.

procédure deelt (in r : radical ;
inout lastm : indxcnst ; inout lastp : indxpr) ;

deelt cherche toutes les décompositions créées à partir du radical *r* et les imprime. *lastr* n'est pas passé en paramètre car les décompositions ne produisent pas de nouveaux radicaux libres.

procédure coelt (in r1, r2 : radical ;
inout lastm : indxcnst ; inout lastp : indxpr) ;

coelt combine deux radicaux libres *r1* et *r2*, et imprime le processus de combinaison obtenu.

Nous donnons les interfaces des deux procédures *rangermol* et *rangerad* qui stockent les constituants produits dans les tableaux *ensmol* et *ensrad*.

procédure rangermol (in m : molécule ;
inout lastm : indxcnst ;
out index : indxcnst) ;

procédure rangerad (in r : radical ;
inout lastr : indxcnst ;
out index : indxcnst) ;

Ces procédures cherchent si le constituant *m* ou *r* est présent dans le tableau correspondant et sinon elle ajoute ce constituant au tableau. *index* est l'indice de *m* ou *r* respectivement dans *ensmol* ou *ensrad*.

2. CAS DES AMORCAGES UNIMOLECULAIRES ET DES ADDITIONS.

L'amorçage unimoléculaire et l'addition d'un radical libre sur une molécule sont les deux modèles de processus simples qui ont un réactif moléculaire, et pour lesquels il faut examiner toutes les liaisons de ce réactif. Cela nécessite un parcours complet de la représentation interne de la molécule. En outre, pour une molécule donnée nous avons un appel unique de la procédure *auelt*, qui génère les amorçages, mais la procédure *adelt*, qui génère les processus d'addition d'un radical donné sur la molécule, est appelée autant de fois qu'il y a de radicaux libres.

Pour éviter un nouveau parcours de la représentation interne d'une molécule à chaque addition d'un nouveau radical sur cette molécule, on introduit un pseudo-constituant, appelé biradical et deux pseudo-modèles de processus p1 et p2.

2.1. Définition d'un biradical et type associé.

Un biradical est un pseudo-constituant contenant deux atomes A et B liés par une liaison simple ou double, et portant chacun un électron célibataire. Le motif d'un biradical est : $\dot{A}—\dot{B}$ ou $\dot{A}=\dot{B}$.

Les deux atomes pointés d'un biradical sont distingués des autres atomes par leur étiquette. On peut donc les considérer comme deux foyers dans le constituant. Un biradical est alors formé de deux arborescences, $A_1 = (X_1, \Gamma_1)$ et $A_2 = (X_2, \Gamma_2)$, dont les racines sont liées par la liaison entre les atomes pointés du biradical. Cette liaison peut être centre de symétrie du biradical, auquel cas les arborescences A_1 et A_2 sont isomorphes, et toutes les classes d'équivalence des atomes du biradical ont un représentant dans X_1. Au contraire, si cette liaison n'est pas centre de symétrie, un élément de X_1 n'est jamais équivalent à un élément de X_2 et réciproquement.

Le type *biradical* associé à la représentation canonique d'un biradical est donc défini par :

```
type biradical = (l : liaison ;
                  cas symque : booléen de
                    vrai : (ar : arbre);
                    faux : (ar1, ar2 : arbre)) ;
```


l est l'étiquette de la liaison entre les deux atomes pointés ;

ar est l'unique représentation canonique des arborescences A_1 et A_2 si la liaison est centre de symétrie ;

ar1 et *ar2* sont les représentations canoniques des arborescences A_1 et A_2 respectivement.

2.2. Les pseudo-modèles de processus p1 et p2.

Le pseudo-modèle de processus p1 crée un biradical en cassant une liaison double ou triple d'une molécule.

$$A=B \longrightarrow \dot{A}—\dot{B}$$

$$\text{ou} \quad A\equiv B \longrightarrow \dot{A}=\dot{B}$$

La liaison $\dot{A}—\dot{B}$ (respectivement $\dot{A}=\dot{B}$) est liaison de symétrie du biradical si et seulement si la liaison $A—B$ (respectivement $A=B$) est liaison de symétrie du biradical.

D'autre part si $A=B$ et $A'=B'$ (respectivement $A\equiv B$ et $A'\equiv B'$) sont deux motifs d'une même molécule, alors les deux pseudo-processus p1 obtenus par rupture des liaisons (A, B) et (A', B') engendrent le même biradical si et seulement si $A \sim A'$ et $B \sim B'$ (ou $A \sim B'$ et $A' \sim B$).

Le pseudo-modèle de processus p2 crée un radical libre en formant une liaison simple entre l'atome pointé d'un radical et un atome pointé d'un biradical.

$$\dot{A}—\dot{B} + \dot{C} \longrightarrow \dot{A}—B—C \quad \text{ou} \quad \dot{B}—A—C$$

$$\dot{A}=\dot{B} + \dot{C} \longrightarrow \dot{A}=B—C \quad \text{ou} \quad \dot{B}=A—C$$

Les deux pseudo-modèles de processus $(p2 : \dot{A}—\dot{B} + \dot{C} \longrightarrow \dot{A}—B—C)$ et $(p2 : \dot{A}—\dot{B} + \dot{C} \longrightarrow \dot{B}—A—C)$ (respectivement $(p2 : \dot{A}=\dot{B} + \dot{C} \longrightarrow \dot{A}=B—C)$ et $(p2 : \dot{A}=\dot{B} + \dot{C} \longrightarrow \dot{B}=A—C)$) sont identiques si et seulement si les deux atomes A et B sont équivalents (cf. § 4.1.1.1. du chapitre III).

L'addition d'un radical libre sur une molécule est alors le bilan de deux processus p1 et p2, tels que le biradical produit par p1 est réactif pour p2. Par exemple le bilan des deux processus suivants :

$$A = B \longrightarrow \dot{A} - \dot{B}$$

et $\dot{A} - \dot{B} + \dot{C} \longrightarrow A - B - C$

fournit le processus d'addition suivant :

$$A = B + \dot{C} \longrightarrow \dot{A} - B - C$$

Exemple :

Considérons l'addition du radical méthyle $\dot{C}H_3$ sur la molécule d'isobutène $CH_2 = C(CH_3)_2$.

Nous appliquons d'abord p1 à la seule liaison double de l'isobutène :

p1 : $CH_2 = C(CH_3)_2 \longrightarrow \dot{C}H_2 - \dot{C}(CH_3)_2$

Puis nous appliquons p2 au biradical obtenu $\dot{C}H_2 - \dot{C}(CH_3)_2$ et au radical $\cdot CH_3$:

p2 : $\dot{C}H_2 - \dot{C}(CH_3)_2 + \dot{C}H_3 \longrightarrow CH_3 - CH_2 - \dot{C}(CH_3)_2$

p2 : $\dot{C}H_2 - \dot{C}(CH_3)_2 + \dot{C}H_3 \longrightarrow \dot{C}H_2 - C(CH_3)_3$

Les processus d'addition sont alors obtenus en faisant le bilan du processus p1 avec chacun des processus p2 :

ad : $CH_2 = C(CH_3)_2 + \dot{C}H_3 \longrightarrow CH_3 - CH_2 - \dot{C}(CH_3)_2$

ad : $CH_2 = C(CH_3)_2 + \dot{C}H_3 \longrightarrow \dot{C}H_2 - C(CH_3)_3$

La décomposition d'un processus d'addition en deux processus de modèles p1 et p2 est unique. Donc deux processus d'addition sont identiques si et seulement si ils définissent le même processus p1 et le même processus p2.

L'introduction des pseudo-processus permet de dissocier la rupture d'une liaison multiple d'une molécule et la formation d'une liaison simple avec un radical libre définies par la sémantique du processus d'addition. L'examen exhaustif des liaisons d'une molécule nécessaire à la recherche des processus d'addition est fait par le pseudo-processus p1, dont la molécule est l'unique réactif. En conservant tous les biradicaux produits par p1 à partir d'une molécule donnée, il devient inutile de parcourir toute la molécule lorsqu'on veut créer des processus d'addition avec un nouveau radical libre. Les biradicaux et les pseudo-modèles de processus n'ont aucun sens chimique et doivent rester transparents à l'utilisateur. Seul le bilan de deux processus p1 et p2, c'est-à-dire le processus d'addition, est imprimé sur la liste résultat.

Pour une molécule donnée, le nombre de biradicaux distincts produits par p1 est égal au nombre de classes d'équivalence des liaisons multiples de la molécule. Inversement tout biradical est associé à une molécule unique. On peut donc stocker les biradicaux dans des listes chaînées, chaque liste étant associée à une molécule. Dans les types *molécule* et *biradical* on ajoute un champ *pb* contenant un pointeur de type *@ biradical* pour construire ces listes chaînées. La définition de ces types est la suivante :

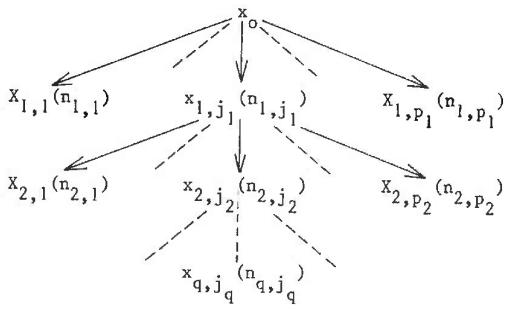
```
type molécule = (pb : @ biradical ;
                 cas symque : booléen de
                   vrai : (moitié : arbrelié) ;
                   faux : (ar : arbre)) ;

type biradical = (pb : @ biradical ; l : liaison ;
                  cas symque : booléen de
                    vrai : (ar : arbre) ;
                    faux : (ar1, ar2 : arbre)) ;
```

La recherche des biradicaux et la construction de la liste chaînée sont faites dans la procédure *auelt* en même temps que la création des processus d'amorçage unimoléculaire. L'interface de cette procédure est légèrement modifiée car le paramètre *m* de type *molécule* qui était un paramètre en entrée est maintenant un paramètre mis à jour. Cette nouvelle interface est la suivante :

```
procédure auelt (inout m : molécule ;
                 inout lastr : indxconst ; inout lastp : indxpr) ;
```

2.3. Amorçage unimoléculaire et pseudo-processus p1.

La représentation canonique d'une molécule non symétrique peut être schématisée par une arborescence X_o de la forme suivante :



où : - chaque noeud $x_{i,j}$ est étiqueté par un symbole atomique ;

- chaque arc $(x_{i,j_i}, x_{i+1,j_{i+1}})$ est étiqueté par le type de la liaison ;

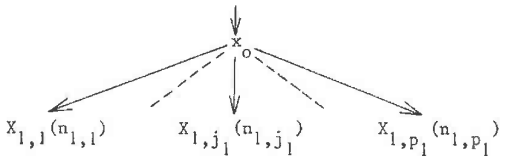
- $X_{i,j}$ est la sous-arborescence de racine $x_{i,j}$;

- n_{i,j_i} est le nombre de substituants identiques, représentés par l'arborescence X_{i,j_i}, du même atome $x_{i-1,j_{i-1}}$.

Il faut noter que le nombre n_{i,j_i} est en général distinct du nombre d'éléments de la classe d'équivalence de x_{i,j_i}. Nous avons par exemple :

$$|\dot{x}_{2,j_2}| = n_{1,j_1} \times n_{2,j_2}$$

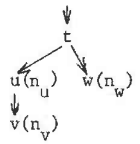
Une molécule symétrique est représentée de façon canonique par un couple (l, X_o) où l est l'étiquette de la liaison de symétrie (liant les deux atomes équivalents x_o) et X_o est une arborescence identique à la précédente. Le schéma d'une molécule symétrique est le suivant :



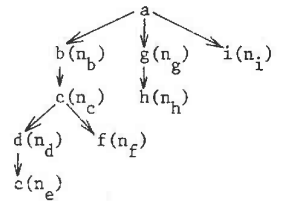
La flèche au-dessus de x_o symbolise la liaison de symétrie (x_o, x_o).

Dans ces représentations canoniques, nous avons un seul représentant pour chaque classe d'équivalence des atomes de la molécule. Donc les amorçages unimoléculaires (respectivement les modèles de processus p1) appliqués aux liaisons simples (respectivement doubles ou triples) explicitées dans les variables de type *arbrelié* qui composent une telle représentation seront tous distincts (cf. § 4.1.1.1. du chapitre III).

Pour créer les processus d'amorçage et les pseudo-processus p1 d'une molécule *m*, nous cherchons d'abord les deux variables *a1* et *a2* de type *arbre* qui définissent les arborescences obtenues en supprimant chaque liaison (x, y) de la représentation canonique de *m*. Les racines de *a1* et *a2* sont les atomes x et y liés par cette liaison (la recherche de *a1* et *a2* est indépendante de la nature des atomes et des liaisons). Les radicaux ou le biradical engendrés par le processus d'amorçage ou le processus p1 sont ensuite construits à partir de *a1*, *a2* et de l'étiquette de la liaison (x, y). Les schémas 1 et 2 ci-dessous montrent tous les couples d'arborescences qu'il faut construire pour la classe de molécules symétriques m1 dont la représentation interne est de la forme :



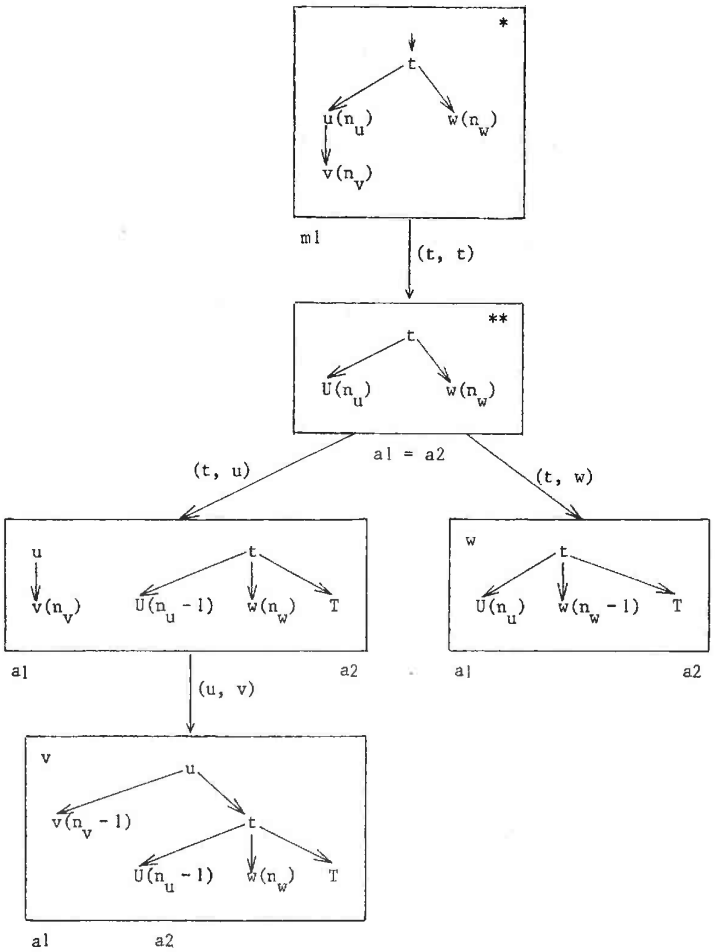
et pour la classe de molécules non symétriques m2 dont la représentation interne est de la forme :



Pour simplifier ces schémas, nous notons X la sous-arborescence de racine x de la représentation interne.

Nous allons commenter ces schémas et en déduire l'algorithme utilisé pour trouver tous ces couples d'arborescences.

Dans une molécule symétrique *m* on cherche d'abord les arborescences associées à la liaison de symétrie, puis on traite toutes les liaisons de



(*) la flèche au-dessus du noeud t symbolise la liaison de symétrie (t, t).

(**) les deux arborescences sont identiques car (t, t) est liaison de symétrie.

Schéma 1 : couples d'arborescences pour la classe des molécules symétriques m1

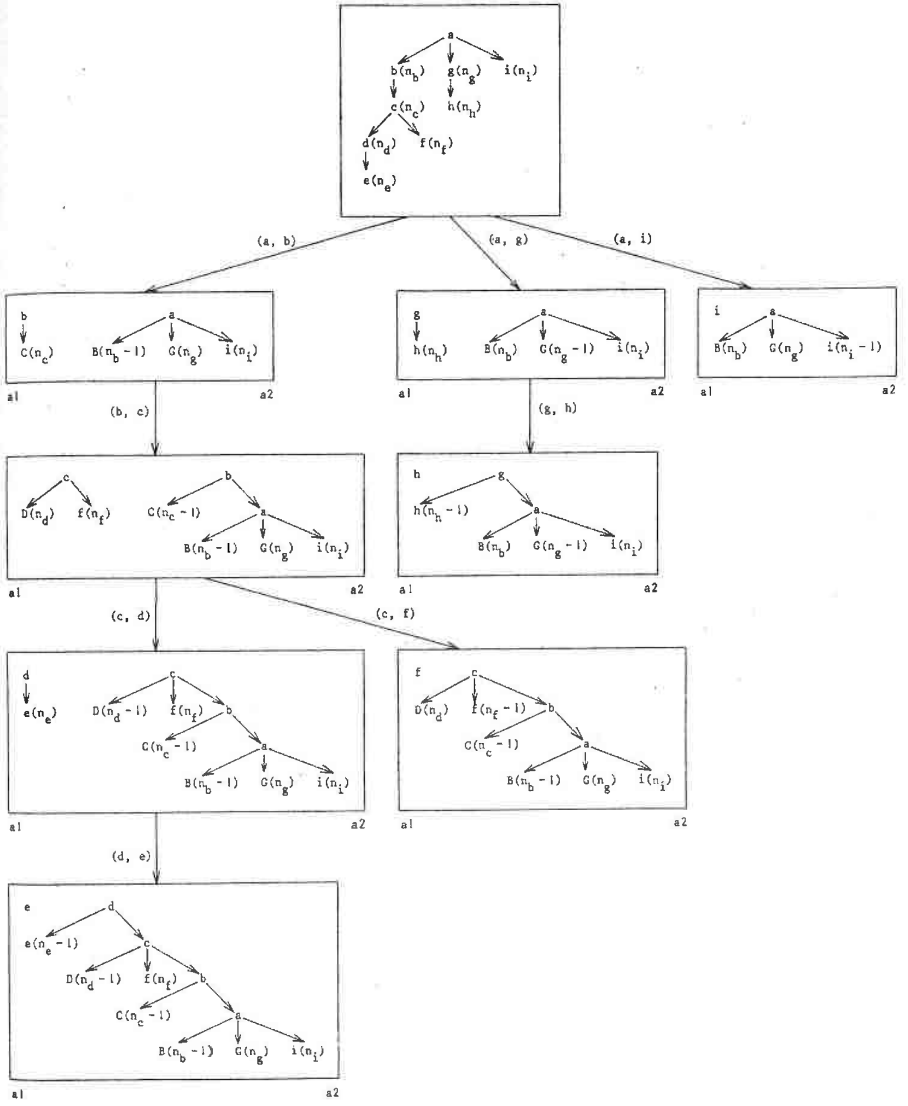


Schéma 2 : couples d'arborescences pour la classe des molécules non symétriques m2

m.moitié.ar dans l'ordre où on les rencontre lorsqu'on parcourt cet arbre "en profondeur d'abord" [FOSDICK 1975]. Par exemple les liaisons de m1 sont traitées dans l'ordre suivant :

(t, t), (t, u), (u, v), (t, w)

Toutes les liaisons d'une molécule *m* non symétrique sont les liaisons de l'arbre *m.ar*. Les liaisons de m2, par exemple, sont traitées dans l'ordre suivant :

(a, b), (b, c), (c, d), (d, e), (c, f), (a, g), (g, h), (a, i)

La procédure *auelt* traite la liaison de symétrie pour une molécule symétrique et appelle une procédure récursive *ausuite* qui traite les liaisons de *m.moitié.ar* si *m* est symétrique ou les liaisons de *m.ar* si *m* n'est pas symétrique. Nous pouvons écrire une première version de ces procédures :

```
procédure auelt (inout  m : molécule ;
                 inout lastr : indxcnst ; inout lastp : indxpr) ;
var a1, a2 : arbre ;
procédure ausuite (in a : arbre) ;
var ax, ay : arbre ;
    i : entier ;
début
  pour i := 1 à a.f.nc faire
    'chercher ax et ay obtenues en supprimant la liaison de
     a.f.descr[i].repr (ay contient l'atome racine de a)' ;
    'en déduire le processus au ou p1' ;
     ausuite (a.f.descr[i].repr.ar) ;
  fpour ;
fin ; % ausuite %
début % auelt %
  si m.symque
    alors
      'chercher a1 et a2 obtenues en supprimant la liaison de
       m.moitié.l (a1 = a2)' ;
      'en déduire le processus au ou p1' ;
       ausuite (m.moitié.ar) ;
```

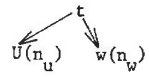


```
    sinon
      ausuite (m.ar) ;
  fsi ;
fin ; % auelt %
```

La construction des arborescences est différente selon que l'on supprime la liaison de symétrie ou les autres liaisons d'une molécule symétrique, les liaisons de l'atome foyer ou les autres liaisons d'une molécule non symétrique.

a) Suppression d'une liaison centre de symétrie (cf. schéma 1).

La suppression de la liaison (t, t) de m1 donne deux arborescences *a1* et *a2* identiques. Leur représentation est la suivante :



Nous avons donc :

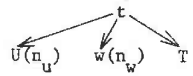
a1 := m1.moitié.ar ; *a2 := a1 ;*

b) Suppression des autres liaisons de m1.

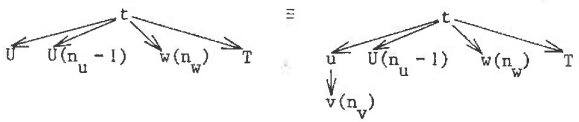
Pour construire les arborescences obtenues en supprimant une liaison de m1 autre que la liaison de symétrie, on cherche d'abord une représentation (non canonique) de m1 dont la racine est l'un des atomes de la liaison. Soit à supprimer par exemple la liaison (u, v) de m1.

Pour obtenir une représentation de m1 de racine u il faut effectuer les opérations suivantes :

- construire une représentation complète de m1, de forme non symétrique, en ajoutant à la racine t de T l'arbrelié qui définit la moitié de m1 :

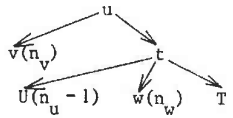


- isoler un substituant U de t :



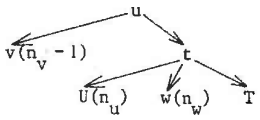
si $n_u = 1$, la forêt $U(n_u - 1) = U(0)$ est la forêt vide ;

- inverser l'orientation de l'arc (t, u) :



La suppression d'une liaison (u, v) revient alors à retirer un substituant v de u. Les arborescences *a1* et *a2* obtenues sont les suivantes :

v et



On peut remarquer que ces deux arborescences *a1* et *a2* peuvent être obtenues plus rapidement à partir de l'étiquette *lxy* de la liaison (t, u) et des arborescences *ax* et *ay* obtenues par suppression de (t, u) (*ax* étant l'arborescence de racine u). En effet

- v est un substituant de la racine u de *ax* ;

- *a2* est obtenue à partir de *ax* en enlevant un substituant v de u et en ajoutant un substituant construit avec la liaison *lxy* et l'arbre *ay*.

L'enlèvement d'un substituant est fait par la

procédure enlever (in s : arbrelié ; inout f : forêt) ;

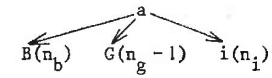
a1 et *a2* sont donc construits par la séquence d'instructions suivantes :

```
% i est un entier de [1, ax.f.nc] %
fx := ax.f ; % on recopie ax.f pour ne pas modifier ax %
a1 := fx.descr[i].repr.ar ;
enlever (i, fx) ; % construction de a2 %
ajouter (1, lier (lxy, ay), fx) ; % fx est la forêt de a2 %
a2 := enraciner (ax.at, fx) ;
% l'étiquette de la liaison supprimée est : ax.f.descr[i].repr.l %
```

c) Suppression des liaisons du foyer d'une molécule non symétrique (cf. schéma 2).

La suppression d'une liaison, (a, g) par exemple, de m2 revient à retirer un des substituants G de a. Les arborescences obtenues sont les suivantes :

G ≡ g (↓ h(n_h)) et

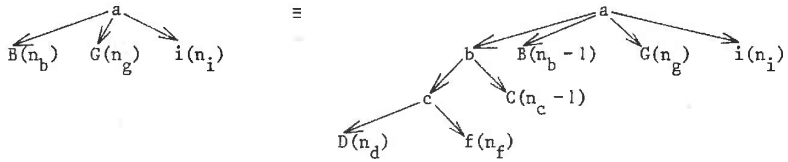


a1 et *a2* sont construits par les instructions suivantes :

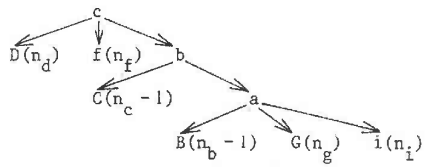
```
% i est un entier de [1, m.ar.f.nc] %
fm := m.ar.f ; % on recopie m.ar.f pour ne pas modifier m %
a1 := fm.descr[i].repr.ar ;
enlever (i, fm) ; % fm est la forêt de a2 %
a2 := enraciner (m.at, fm) ;
% l'étiquette de la liaison supprimée est : m.f.descr[i].repr.l %
```

d) Suppression des autres liaisons de m1.

Comme dans (b), les arborescences obtenues en supprimant une autre liaison que celles de l'atome foyer sont construites à partir d'une représentation (non canonique) de la molécule dont la racine est l'un des atomes foyers de la liaison. Par exemple pour la liaison (c, d) de m1 on construit la représentation de racine c.

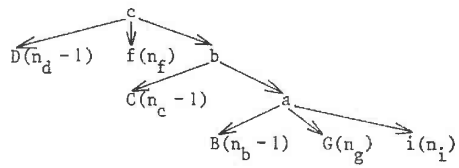


On inverse l'orientation des arêtes (a, b) et (b, c).



Les arborescences *a1* et *a2* obtenues sont les suivantes :

D ≡ d (↓ e(n_e)) et



a1 et *a2* peuvent être construites directement à partir de l'étiquette *lxy* de la liaison (b, c) et des arborescences *ax* et *ay* obtenues en supprimant (b, c) avec les mêmes instructions qu'en (b).

Nous avons donc trois traitements distincts pour supprimer les liaisons d'une molécule correspondant à :

(1) la liaison de symétrie d'une molécule symétrique (a) ;

(2) les liaisons de l'atome foyer d'une molécule non symétrique (c) ;

(3) les autres liaisons (b et c).

Nous modifions les spécifications des procédures *auelt* et *ausuite*

- *auelt* traite les cas (1) et (2)
- *ausuite* traite le cas (3).

Les paramètres en entrée nécessaires à *ausuite* sont :

- *lxy* de type *liaison*
- *ax* et *ay* de type *arbre*

Avant de donner une nouvelle version de ces procédures nous raffinons l'action "en déduire le processus au ou p1".

A ce niveau la molécule est définie par deux arbres *ax* et *ay* de racines x et y et l'étiquette *lxy* de la liaison (x, y) de la molécule. Si (x, y) est une liaison simple, sa rupture engendre deux radicaux libres *pointer (ax)* et *pointer (ay)* qu'il faut ajouter à l'ensemble *ensrad*. La création du processus d'amorçage unimoléculaire :

m ⟶ *pointer (ax) + pointer (ay)*

c'est-à-dire son impression et son stockage dans le tableau *ensproc*, est faite par la

procédure créerau (in m : molécule ; in r1, r2 : radical ;
inout lastp : indxpr) ;

Si la liaison (x, y) est double ou triple, sa rupture engendre un biradical construit avec la liaison *pred (lxy)* et les deux arbres *ax* et *ay*. Ce biradical est inséré à la fin de la chaîne pointée par *m.pb*. Seule la rupture d'une liaison de symétrie engendre un biradical symétrique. Il est alors inséré en tête de chaîne. On ne crée pas de processus p1 car c'est un pseudo-processus qui est transparent à l'utilisateur.

Ce traitement est fait par une procédure *aup1* dont l'interface est :

procédure aup1 (in lxy : liaison ; in ax, ay : arbre ;
inout q : @ biradical) ;

A l'entrée de *aup1*, le pointeur *q* est égal à *nil* si la chaîne *m.pb* est vide, sinon *q* pointe son dernier élément. A la fin de *aup1*, *q* est inchangé si *lxy* est simple, pointe le nouveau biradical si *lxy* est double ou triple.

Comme *aup1* est appelée par *ausuite*, il faut faire transiter le pointeur *q* par cette procédure, c'est-à-dire que *q* est un nouveau paramètre mis à jour de *ausuite*. Cependant le traitement d'une liaison de symétrie est fait par la procédure *auelt*.

Nous pouvons maintenant donner l'algorithme de ces trois procédures :

```
procédure auelt (inout m : molécule ;
                 inout lastr : radical ; inout lastp : indxpr) ;
var i : entier ;
    a1, a2 : arbre ;
    fm : forêt ;
    p : @ biradical ;
    ia1 : indxcnst ;
procédure aup1 (in lxy : liaison ; in ax, ay : arbre ;
                inout q : @ biradical) ;
var ix, iy : indxcnst ;
début
  si lxy = simple
    alors % processus au %
      rangerad (pointer (ax), lastr, ix) ;
      rangerad (pointer (ay), lastr, iy) ;
      créerau (m, pointer (ax), pointer (ay), lastp) ;
    sinon % pseudo-processus p1 %
      si q = nil
        alors % la liste est vide %
          allouer (m.pb) ; q := m.pb ;
        sinon
          allouer (q@ .pb) ; q := q@ .pb ;
      fsi ;
```

```
      q@ .symque := faux ; % lxy n'est pas liaison de symétrie %
      q@ .l := pred (lxy) ;
      q@ .ar1 := ax ; q@ .ar2 := ay ;
      q@ .pb := nil ;
  fsi ;
fin ; % aup1 %

procédure ausuite (in lxy : liaison ; in ax, ay : arbre ;
                   inout q : @ biradical) ;
var i : entier ;
    a1, a2 : arbre ;
    fx : forêt ;
début
  pour i := 1 à ax.f.nc faire
    % on supprime la liaison des substituants i de ax.at %
    fx := ax.f ; % on recopie ax.f pour ne pas modifier ax %
    a1 := fx.descr[i].repr.ar ;
    enlever (i, fx) ; % construction de a2 %
    ajouter (1, lier (lxy, ay), fx) ; % fx est la forêt de a2 %
    a2 := enraciner (ax.at, fx) ;
    aup1 (ax.f.descr[i].repr.l, a1, a2, q) ;
    % suppression des liaisons de a1 %
    ausuite (ax.f.descr[i].repr.l, a1, a2, q) ;
  fpour ;
fin ; % ausuite %

début % auelt %
  p := nil ; m.pb := nil ;
  si m.symque
    alors % suppression de la liaison de symétrie %
      a1 := m.moitié.ar ; % a2 = a1 %
      si m.moitié.l = simple
        alors % processus au %
          rangerad (pointer (a1), lastr, ia1) ;
          créerau (m, pointer (a1), pointer (a1), lastp) ;
        sinon % pseudo-processus p1 %
```



```
          allouer (m.pb) ; p := m.pb ;
          p@ .symque := vrai ;
          p@ .l := pred (m.moitié.l) ;
          p@ .ar := a1 ;
          p@ .pb := nil ;
      fsi ;
      ausuite (m.moitié.l, a1, a1, p) ; % suppression des autres
                                          liaisons %
    sinon % m non symétrique %
      pour i := 1 à m.ar.f.nc faire
        % suppression de la liaison des substituants i de m.at %
        fm := m.f ; % on recopie m.f pour ne pas modifier m %
        a1 := fm.descr[i].repr.ar ;
        enlever (i, fm) ; % fm est la forêt de a2 %
        a2 := enraciner (m.at, fm) ;
        aup1 (m.ar.f.descr[i].repr.l, a1, a2, p) ;
        % suppression des liaisons de a1 %
        ausuite (m.ar.f.descr[i].repr.l, a1, a2, p) ;
      fpour ;
  fsi ;
fin ; % auelt %
```

2.4. Addition d'un radical libre sur une molécule.

Comme nous l'avons dit au paragraphe 1.2. l'addition d'un radical libre sur une molécule est le bilan d'un pseudo-processus p1 qui crée un biradical b à partir d'une molécule m et d'un pseudo-processus p2 qui forme une liaison simple entre l'atome pointé d'un radical r et un des atomes pointés du biradical b. Un biradical symétrique engendre un seul pseudo-processus p2 avec un radical donné r, et un biradical non symétrique en engendre deux. Les biradicaux associés à une molécule m sont rangés dans une liste chaînée pointée par $m.pb$ (cf. § 1.2.). Un biradical symétrique est obtenu uniquement si la molécule associée est symétrique et si sa liaison de symétrie est double ou triple. Ce biradical est alors rangé à la tête de la chaîne $m.pb$.

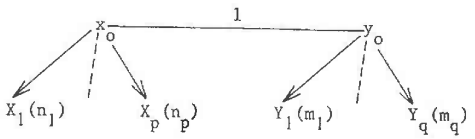
Comme pour les amorçages la création d'un processus d'addition est faite par une procédure *créerad* dont l'interface est :

```
procédure créerad (in m : molécule ; in r, rad : radical ;
                   inout lastp : indxpr) ;
```

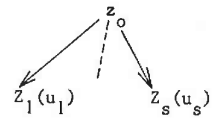
Nous donnons une première version de l'algorithme de *adelt*.

```
procédure adelt (in m : molécule, in r : radical ;
                 inout lastr : indxconst ; inout lastp : indxpr) ;
var p : @ biradical ;
    b : biradical ;
    rad1, rad2 : radical ;
    irad1, irad2 : indxconst ;
début
  p := m@ .pb ;
  si m.symque etsi m.moitié.l ⩾ double
    alors % on a un biradical symétrique %
      'construire rad1 avec le biradical symétrique pointé par p
       et le radical r' ;
       rangerad (rad1, lastr, irad1) ;
       créerad (m, r, rad1, lastp) ;
       p := p@.pb ;
  fsi ;
  % traitement des biradicaux non symétriques %
    tantque p ≠ nil faire
      'construire rad1 et rad2 avec le biradical non symétrique
       pointé par p et le radical r' ;
       rangerad (rad1, lastr, irad1) ;
       rangerad (rad2, lastr, irad2) ;
       créerad (m, r, rad1, lastp) ;
       créerad (m, r, rad2, lastp) ;
       p := p@ .pb ;
    ftantque ;
fin ; % adelt %
```

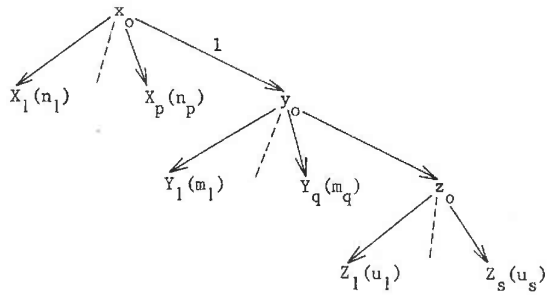
La représentation canonique d'un biradical non symétrique est :

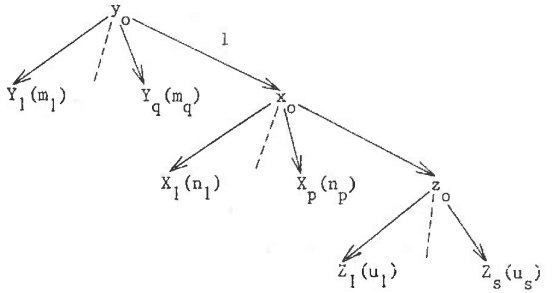


Celle d'un radical est :



Le pseudo-modèle de processus p2 engendre un radical de foyer x_o en formant une liaison simple entre y_o et z_o, et un radical de foyer y_o en formant une liaison simple entre x_o et z_o. Ces radicaux ont la forme suivante :





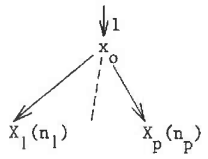
Pour construire *rad1*, par exemple, il faut donc :

- recopier *b.ar2.f* dans une variable auxiliaire *f1* de type *forêt*, pour ne pas modifier *b* ;
  % $f_1 = \{Y_1(m_1), \ldots Y_q(m_q)\}$ %

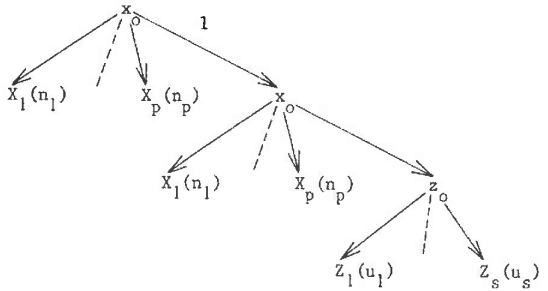
- ajouter à *f1* le substituant *lier (simple, r.ar)* ;
  % *f1* = $\{Y_1(m_1), \ldots Y_q(m_q), Z_o\}$ %
- recopier *b.ar1.f* dans une variable auxiliaire *f2* de type *forêt* ;
  % *f2* = $\{X_1(m_1), \ldots X_p(m_q)\}$ %
- ajouter à *f2* le substituant *lier (l, enraciner (ar2.at, f1))* ;
- *enraciner (ar1.at, f2)*.

On obtient *rad2* de la même façon en échangeant *ar1* et *ar2*.

Pour un biradical symétrique, dont la représentation est :



le radical unique obtenu est



La construction du radical est identique dans ce cas, mais il faut remplacer *ar1* et *ar2* par *ar*.

Les radicaux obtenus par le processus p2 sont construits par une procédure *p2* qui a trois paramètres en entrée :

- *l* de type *liaison* (liaison du biradical) ;
- *arx* et *ary* de type *arbre* (arbres du biradical) ;

et un paramètre *rxy* en sortie de type *radical*.

La procédure *p2* est interne à *adelt*.

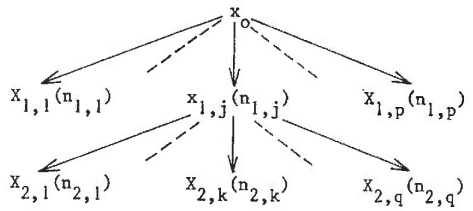
L'algorithme complet de *adelt* est le suivant :

```
procédure adelt (in m : molécule ; in r : radical ;
                 inout lastr : indxcnst ; inout lastp : indxpr) ;
var p : @ biradical ;
    b : biradical ;
    rad1, rad2 : radical ;
    irad1, irad2 : indxcnst ;
procédure p2 (in l : liaison ; in arx, ary : arbre ;
              out rxy : radical) ;
var f1, f2 : forêt ;
début
  f1 := b.ary.f ;
  ajouter (1, lier (simple, r.ar), f1) ;
  f2 := b.arx.f ;
  ajouter (1, lier (l, enraciner (ary.at, f1), f2) ;
  rxy := pointer (enraciner (arx.at, f2)) ;
fin ; % p2 %

début % adelt %
  p := m@ .pb ;
  si m.symque etsi m.moitié.l ⩾ double
    alors % on a un biradical symétrique %
      % construire rad1 avec le biradical symétrique pointé par
      p et le radical r %
      p2 (b.l, b.ar, b.ar, rad1) ;
      rangerad (rad1, lastr, irad1) ;
      créerad (m, r, rad1, lastp) ;
      p := p@ .pb ;
  fsi ;
  % traitement des biradicaux non symétriques %
  tantque p ≠ nil faire
    % construire rad1 et rad2 avec le biradical non symétrique
    pointé par p et le radical r %
    p2 (b.l, b.ar1, b.ar2, rad1) ;
    p2 (b.l, b.ar2, b.ar1, rad2) ;
```

```
        rangerad (rad1, lastr, irad1) ;
        rangerad (rad2, lastr, irad2) ;
        créerad (m, r, rad1, lastp) ;
        créerad (m, r, rad2, lastp) ;
        p := p@ .pb ;
    ftantque ;
fin ; % adelt %
```

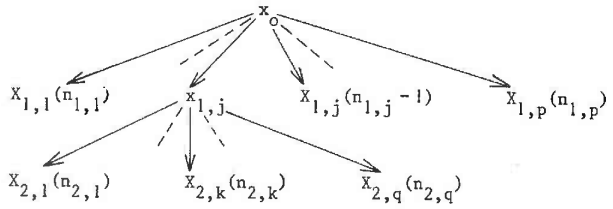

3. DECOMPOSITION D'UN RADICAL LIBRE.

Soit r un radical dont la représentation canonique est :

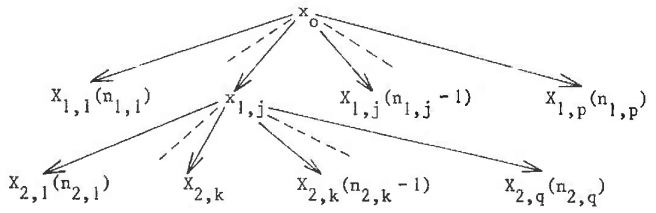


Si la liaison $(x_o, x_{1,j})$ est simple ou double, et si la liaison $(x_{1,j}, x_{2,k})$ est simple, on décompose le radical en rompant la liaison $(x_{1,j}, x_{2,k})$. Ce processus engendre un radical *rde* et une molécule *m*. La représentation canonique de *rde* est $X_{2,k}$. Pour construire une représentation arborescente *ra* de *m*, non canonique en général, il faut :

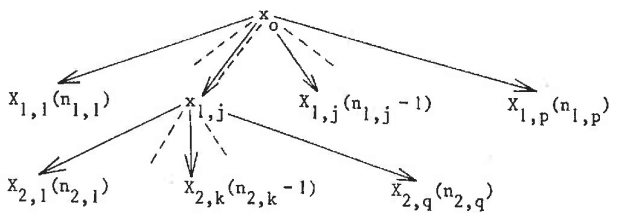
- isoler un substituant $X_{1,j}$ de x_o :



- isoler un substituant $X_{2,k}$ de $x_{1,j}$:



- enlever le substituant $X_{2,k}$ et augmenter la multiplicité de la liaison $(x_o, x_{1,j})$, ce que nous symbolisons par un trait pointillé.



Soient s_j la variable de type *arbrelié* associée à $X_{1,j}$ et *sk* la variable de type *arbrelié* associée à $X_{2,k}$. Le radical *rde* est obtenu simplement par :

rde := pointer (sk.ar) ;

Pour construire *ra* il faut :

- copier *r.ar.f* dans une variable *fj* de type *forêt* ;

$$\% \; fj = \{X_{1,1}(n_{1,1}),\ldots,X_{1,j}(n_{1,j}),\ldots,X_{1,p}(n_{1,p})\} \;\%$$

- enlever un substituant *sj* de *fj* ;

$$\% \; fj = \{X_{1,1}(n_{1,1}),\ldots,X_{1,j}(n_{1,j}-1),\ldots,X_{1,p}(n_{1,p})\} \;\%$$

- copier *sj.ar.f* dans une variable *fk* de type *forêt* ;

$$\% \; fk = \{X_{2,1}(n_{2,1}),\ldots,X_{2,k}(n_{2,k}),\ldots X_{2,q}(n_{2,q})\} \;\%$$

- enlever un substituant *sk* de *fk* ;

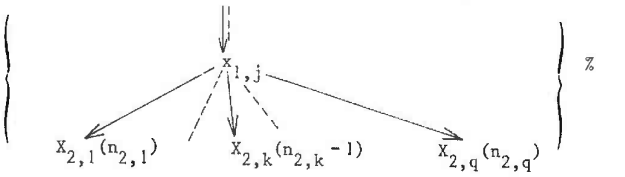
$$\% \; fk = \{X_{2,1}(n_{2,1}),\ldots,X_{2,k}(n_{2,k}-1),\ldots,X_{2,q}(n_{2,q})\} \;\%$$

- ajouter à *fj* le substituant ;

lier (succ (sj.l), enraciner (sj.ar.at, fk)) ;

$$\% \; fj = \{X_{1,1}(n_{1,1}),\ldots,X_{1,j}(n_{1,j}-1),\ldots,X_{1,j}(n_{1,p})\}$$

$$+ \left\{ \quad \right. \quad \left. \right\} \;\%$$



- enraciner *fj* à *r.ar.at* ;

La molécule *m* engendrée par cette décomposition est alors la représentation canonique de *ar* obtenue en appelant la procédure *canonique* définie au chapitre précédent (§ 4.3).

L'interface de la procédure *créerde* est :

```
procédure créerde (in r, rde : radical ; in m : molécule ;
                   inout lastp : indxpr) ;
```

L'algorithme de *deelt* est le suivant :

```
procédure deelt (in r : radical ;
                 inout lastm : indxcnst ; inout lastp : indxpr) ;

var sj, sk : arbrelié ;
    fj, fk : forêt ;
    j, k   : entier ;
    rde    : radical ;
    m      : molécule ;
    ra     : arbre ;
    im     : indxcnst ;

début
  pour j := 1 à r.ar.f.nc faire
    sj := r.ar.f.descr[j].repr ;
    si sj.l ≤ double  % liaison (x_o, x_1,j) %
      alors
        pour k := 1 à sj.ar.f.nc faire
          sk := sj.ar.f.descr[k].repr ;
          si sk.l = simple  % liaison (x_1,j, x_2,k) %
            alors % on rompt la liaison sk.l %
              % construction de rde %
              rde := pointer (sk.ar) ;
              % construction de ra %
              fj := r.ar.f ; enlever (j, fj) ;
              fk := sj.ar.f ; enlever (k, fk) ;
              ajouter (1, lier (succ(sj.l),
                                enraciner (sj.ar.at,fk)), fj) ;
              ra := enraciner (r.ar.at, fj) ;
              % recherche de la forme canonique m de ra %
              canonique (ra, m) ;
              % création du processus de décomposition %
              rangermol (m, lastm, im) ;
              créerde (r, rde, m, lastp) ;
```

```
            fsi ;
          fpour ;
        fsi ;
      fpour ;
    fin ; % deelt %
```

Remarque :

Il faut copier la variable $r.ar.f$ dans fj au début de chaque boucle k, sinon on accumule dans fj tous les nouveaux substituants qu'on y ajoute.

4. COMBINAISON DE DEUX RADICAUX LIBRES.

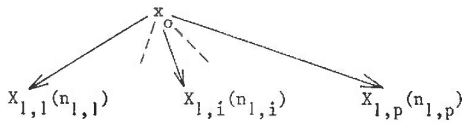
4.1. Cas de deux radicaux identiques.

La combinaison de deux radicaux r identiques produit une molécule dont la liaison formée est liaison de symétrie. Le foyer du radical est donc un foyer double pour la molécule. Nous avons donc immédiatement la fonction de construction de la variable m de type *molécule* à partir de la variable r de type *radical*

```
m.symque := vrai ;
m.moitié := lier (simple, r.ar) ;
```

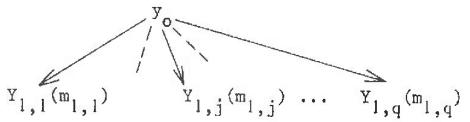
4.2. Cas de deux radicaux distincts.

Soient r et s deux variables de type *radical*.
Les représentations canoniques sont :



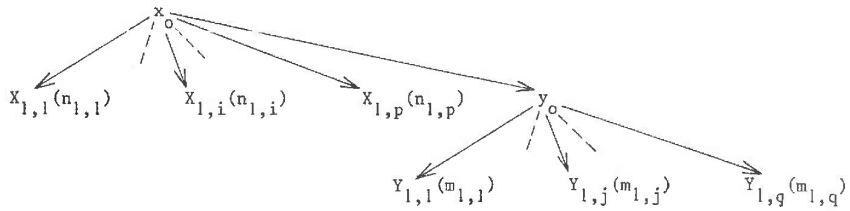
pour r

et



pour s

La molécule m est obtenue en formant une liaison simple entre x_o et y_o.
Une des représentations arborescentes ra de m est par exemple :



Cette représentation n'est pas canonique en général.

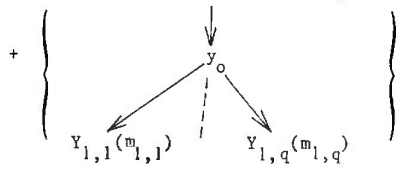
Pour construire *ra* il faut :

- copier *r.ar.f* dans une variable *fr* de type *forêt* ;

% *fr* = $\{X_{1,1}(n_{1,1}),\ldots,X_{1,p}(n_{1,p})\}$ %

- ajouter à *fr* le substituant *lier (simple, s.ar)* ;

% *fr* = $\{X_{1,1}(n_{1,1}),\ldots X_{1,p}(n_{1,p})\}$

+ 

%

- *enraciner fr* à l'atome *r.ar.at* ;

La molécule engendrée par la combinaison est la forme canonique de *ra*.

L'interface de la procédure *créerco* est :

```
procédure créerco (in r1, r2 : radical ; in m : molécule ;
                   inout lastp : indxpr) ;
```

L'algorithme de *coelt* est donc :

```
procédure coelt (in r, s : radical ;
                 inout lastm : indxcnst ; inout lastp : indxpr) ;
var ra : arbre ;
    fr : forêt ;
    im : molécule ;

début
  si r = s
    alors
      m.symque := vrai ;
      m.moitié := lier (simple, r.ar) ;
    sinon
      % construction de ra %
      fr := r.ar.f ;
      ajouter (1, lier (simple, s.ar), fr) ;
      ra := enraciner (r.ar.at, fr) ;
```



```
      % recherche de la forme canonique de ra %
      canonique (ra, m) ;
  fsi ;
  rangermol (m, lastm, im) ;
  créerco (r, s, m) ;
fin ; % coelt %
```

Les quatre procédures *auelt*, *adelt*, *deelt*, *coelt* décrivent l'application des modèles de processus simples aux constituants. Nous avons maintenant résolu les principaux problèmes posés par la construction automatique de mécanismes réactionnels, à l'exception de la création des processus composés. Mais comme nous l'avons vu au paragraphe 3.4.1 du chapitre II, ceux-ci sont obtenus par superposition de deux processus simples, et leur recherche est davantage un problème d'implantation, qu'un problème de description algorithmique. Celle-ci est traitée au chapitre suivant.

CHAPITRE V

Implantation

Dans les chapitres précédents, nous avons décrit notre application dans un langage algorithmique. Nous allons maintenant préciser certains points propres à une implantation de cette application en langage PASCAL [WIRTH 1971], [MAURICE 1978]. Ces points sont les suivants :

- définition des types PASCAL pour les objets abstraits ;
- comparaison des objets abstraits ;
- analyse syntaxique des constituants ;
- stockage des processus ;
- recherche des processus composés.

1. DEFINITION DES TYPES PASCAL POUR LES OBJETS ABSTRAITS.

Dans le chapitre III nous avons défini les objets abstraits suivants :

```
arbre = (at : atome ; f : forêt) ;
forêt = (nc : entier ; descr : tableau [1..nc] de jumeaux) ;
jumeaux = (repr : arbrelié ; nb : entier) ;
arbrelié = (l : liaison ; ar : arbre) ;
molécule = (pb : @ biradical ;
            cas symque : booléen de
              vrai : (moitié : arbrelié) ;
              faux : (ar : arbre)) ;
radical  = (ar : arbre) ;
biradical = (pb : @ biradical ;
              cas symque : booléen de
                vrai : (ar : arbre) ;
                faux : (ar1, ar2 : arbre)) ;
```

Ces objets ne peuvent être implantés directement en PASCAL pour les raisons suivantes :

- le nombre d'éléments du tableau *descr*, défini dans l'objet *forêt*, est variable ;

- les définitions des objets *arbre*, *forêt*, *jumeaux*, *arbrelié* sont récursives.

1.1. Majoration du nombre d'éléments d'un objet de type *forêt*.

Pour supprimer la dimension variable du tableau *descr* d'un objet de type *forêt*, il faut majorer son nombre d'éléments. Cet objet a été défini pour construire la liste des substituants d'un atome dans un constituant. Il ne peut donc pas contenir plus d'éléments que la valence d'un atome. La taille du tableau *descr* est alors majorée par la plus grande valence des atomes. Cette valeur est donnée par une constante *maxval* que nous avons fixée à 4 pour des applications aux hydrocarbures halogénés. Le type *forêt* est défini par :


```
forêt = struct
          nc : 0..maxval ;
          descr : tableau [1..maxval] de jumeaux ;
        fin ;
```

Pour une variable f de type *forêt*, les éléments $f.descr.[i]$, i de $nc+1$ à *maxval* sont vides, ce que nous indiquerons en initialisant à zéro le champ *nb* de ces éléments. Nous avons donc :

$$\forall i \in [nc+1,\ maxval],\ f.descr[i].nb = 0$$

Le champ *nc* d'une forêt vide est nul, ce qui est le cas, en particulier, pour la variable *fvide*.

1.2. Rupture de la boucle de récursivité dans la définition des objets *arbre*, *arbrelié*, *forêt*.

Pour rompre les récursivités dans les définitions des objets, nous avons choisi de stocker dans un tableau *enssubst*, de taille *maxsubst* tous les objets *arbrelié* nécessaires à la construction des molécules et des radicaux libres. Nous avons donc :

```
var enssubst : tableau [1..maxsubst] de arbrelié ;
```

avec par exemple :

```
const maxsubst = 200 ;
```

Nous définissons aussi le

```
type indxsubst = 0..maxsubst ;
```

Le dernier élément de *enssubst* est repéré par la variable

```
var lastsubst : indxsubst ;
```

lastsubst est initialisée à zéro ce qui exprime que le tableau est vide.

A chaque fois qu'un objet de type *arbrelié* est construit (lors de la lecture des réactifs, ou de la création des processus élémentaires), la

```
procédure rangersubst (in al : arbrelié ; inout lastsubst : indxsubst ;
                       out index : indxsubst) ;
```

est appelée. Celle-ci cherche d'abord si la variable *al* est dans le tableau *enssubst*. Si elle s'y trouve, *rangersubst* retourne dans le paramètre *index*,

l'indice de *al* dans le tableau (*lastsubst* est alors inchangée) ; sinon *rangersubst* range à la fin du tableau la variable *al* et met à jour *lastsubst* ; *index* a alors la même valeur que *lastsubst*. Deux éléments du tableau *enssubst* contiennent donc toujours deux objets de type *arbrelié* distincts.

Nous modifions en conséquence l'objet *jumeaux* qui fait référence à un *arbrelié*.

```
jumeaux = struct
            repr : 0..maxsubst ;
            nb : entier ;
          fin ;
```

La valeur zéro du champ *repr* est utilisée pour une occurrence vide de l'objet *jumeaux*. Dans ce cas le champ *nb* vaut aussi zéro.

Exemple

Nous donnons à titre d'exemple le tableau *enssubst* nécessaire au stockage des substituants suivants :

s_1 : $-C(CH_3)_2-CH_3$, s_2 : $=C(CH_2-CH_3)(CH_3)$, s_3 : $=CH-C(CH_3)_2-CH_3$

a) s_1 est construit avec le substituant $-CH_3$ qui lui-même est construit à partir de $-H$.
Donc *enssubst*[*1*] contient $-H$, *enssubst*[*2*] contient $-CH_3$ et *enssubst*[*3*] contient s_1.

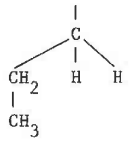
b) s_2 est construit à partir de $-CH_3$ qui est déjà dans *enssubst*. Mais avant de ranger s_2, il faut construire $-CH_2-CH_3$.

c) s_3 est construit à partir de s_1 et de $-H$.

Pour plus de clarté, nous avons laissé le symbole atomique de chaque atome, au lieu de leur code lexicographique.

	l	*ar*						substituant
		at	*nc*	*f*				
				descr[*1*]	*descr*[*2*]	*descr*[*3*]	*descr*[*4*]	
				repr (×*nb*)	*repr* (×*nb*)	*repr* (×*nb*)	*repr* (×*nb*)	
1	/	H	0	0 (×0)	0 (×0)	0 (×0)	0 (×0)	/H
2	/	C	1	1 (×3)	0 (×0)	0 (×0)	0 (×0)	/CH3
(s_1) 3	/	C	1	2 (×3)	0 (×0)	0 (×0)	0 (×0)	/C(CH3)3
4	/	C	2	2 (×1)	1 (×2)	0 (×0)	0 (×0)	/CH2/CH3
(s_2) 5	//	C	2	4 (×1)	2 (×1)	0 (×0)	0 (×0)	//C(CH3)(CH2/CH3)
(s_3) 6	//	C	2	3 (×1)	1 (×1)	0 (×0)	0 (×0)	//CH/C(CH3)3

Le substituant *enssubst*[*4*], par exemple, est constitué d'une liaison simple et d'un atome C ; cet atome a un substituant identique à *enssubst*[*2*] (référencé par *enssubst*[*4*].*ar.f.descr*[*1*]) et deux substituants identiques à *enssubst*[*1*] (référencés par *enssubst*[*4*].*ar.f.descr*[*2*]) ; *enssubst*[*4*] a donc la forme suivante :



Remarques

1) Tous les objets *arbrelié* référencés par un élément *enssubst*[*i*].*f* ont un indice strictement inférieur à *i*, c'est-à-dire que nous n'avons pas de référence en avant.

2) Rappelons que dans l'objet *forêt*, les représentants du tableau *descr* sont classés dans l'ordre décroissant :

- les champs *ar.f* de *enssubst*[*1*], *enssubst*[*2*], *enssubst*[*3*] n'ont qu'un seul représentant ;

- pour *enssubst*[*4*], il faut comparer *enssubst*[*1*] et *enssubst*[*2*] :

nous avons :

complexité (enssubst[1].ar) = 1
complexité (enssubst[2].ar) = 2

donc :

enssubst[1] < enssubst[2] ;

- pour *enssubst[5]*, il faut comparer *enssubst[2]* et *enssubst[4]* : les champs *ar* de ces deux éléments ont même complexité (égale à 2), mais nous avons :

enssubst[2].ar.f.descr[1].repr < enssubst[4].ar.f.descr[1].repr

donc :

enssubst[2] < enssubst[4]

- pour *enssubst[6]*, il faut comparer *enssubst[1]* et *enssubst[3]* :

complexité (enssubst[1].ar) = 1
complexité (enssubst[3].ar) = 3

donc :

enssubst[1] < enssubst[3]

Dans les deux tableaux qui suivent nous avons représenté quelques molécules et quelques radicaux libres que nous pouvons construire avec ces substituants. Pour les molécules, nous avons omis le champ *pb*, et nous avons fait figurer les trois champs *symque*, *ar* et *moitié* :

si *symque = faux*, le champ *ar* est utilisé ;
si *symque = vrai*, le champ *moitié* est utilisé.

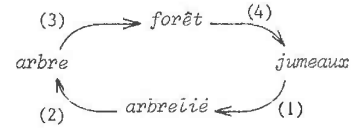
symque	*ar*						*moitié*	molécule
	at	*f*						
		nc	*descr[1]*	*descr[2]*	*descr[3]*	*descr[4]*		
			repr (×nb)	*repr (×nb)*	*repr (×nb)*	*repr (×nb)*		
faux	C	1	1 (×4)	0 (×0)	0 (×0)	0 (×0)		CH4
vrai							2	/(CH3)2
faux	C	3	4 (×1)	2 (×2)	1 (×1)	0 (×0)		CH(CH3)2/CH2/CH3

représentation de quelques molécules

at	*f*					radical libre
	nc	*descr[1]*	*descr[2]*	*descr[3]*	*descr[4]*	
		repr (×nb)	*repr (×nb)*	*repr (×nb)*	*repr (×nb)*	
H	0	0 (×0)	0 (×0)	0 (×0)	0 (×0)	.H
C	1	1 (×3)	0 (×0)	0 (×0)	0 (×0)	.CH3
C	2	4 (×1)	2 (×2)	0 (×0)	0 (×0)	.C(CH3)2/CH2/CH3
C	2	6 (×1)	1 (×1)	0 (×0)	0 (×0)	.CH//CH/C(CH3)3

représentation de quelques radicaux libres

Nous aurions pu rompre la boucle de récursivité :



en (2), (3) ou (4). Mais les coupures en (3) ou (4) entraînent des références à des objets *forêt* ou *jumeaux* qui n'ont pas de sens chimique propre. Le stockage des objets *arbre* requis par une coupure en (2) simplifiait la structure interne des variables de type *molécule* et *radical*. Mais tous les objets *arbre* ainsi construits, comme par exemple CH, CH2, CH / CH3, n'ont pas de sens chimique.

Pour supprimer la récursivité dans la définition des objets nous aurions pu également utiliser des pointeurs. Mais nous avons préféré construire un tableau et accéder à ses éléments par leur indice.

1.3. Récapitulation de la définition des types PASCAL.

Nous venons de donner les modifications nécessaires à la définition des objets pour les implanter en PASCAL. En outre nous avons vu que la complexité définie pour un objet de type *arbre* était souvent utilisée, pour classer les substituants dans une *forêt*, et pour chercher le foyer d'une molécule. Au lieu de recalculer à chaque fois cette complexité, ce qui impose un parcours complet de l'arbre, nous la stockons dans un nouveau champ *cplx* de type *entier* ajouté au type *arbre* :

```
type arbre = struct
               cplx : entier ;
               at : atome ;
               f : forêt ;
             fin ;
```

Nous récapitulons maintenant l'ensemble de tous les types PASCAL associés aux objets :

```
const maxval, maxsubst, maxatom ;

type  liaison = (simple, double, triple) ;
      atome = 1..maxatom ;
      indxsubst = 0..maxsubst ;
      jumeaux = struct
                  repr : indxsubst ;
                  nb : 0..maxval ;
                fin ;
      forêt   = struct
                  nc : 0..maxval ;
                  descr : tableau [1..maxval] de jumeaux ;
                fin ;
      arbre   = struct
                  cplx : entier ;
                  at : atome ;
                  f : forêt ;
                fin ;
```



```
arbrelié = struct
             l : liaison ;
             ar : arbre ;
           fin ;
molécule = struct
             pb : @ biradical ;
             cas symque : booléen de
               vrai : (moitié : arbrelié) ;
               faux : (ar : arbre) ;
           fin ;
radical  = arbre ;
biradical= struct
             pb : @ biradical ;
             cas symque : booléen de
               vrai : (ar : arbre) ;
               faux : (ar1, ar2 : arbre) ;
           fin ;
```

Remarque :

Le champ *nb* du type *jumeaux* est majoré par *maxval* pour les mêmes raisons que le champ *nc* du type *forêt*.

Les définitions de ces types ne sont pas récursives car le champ *repr* de *jumeaux* désigne un indice du tableau *enssubst*, et non plus directement un objet de type *arbrelié*.

2. RELATION D'ORDRE SUR LES OBJETS DE TYPE *ARBRELIE*.

2.1. Position du problème.

La relation d'ordre sur les objets de type *arbrelié* a été définie au paragraphe 4.2 du chapitre III. Celle-ci est utilisée pour classer les substituants d'un atome donné dans un constituant. La comparaison de deux variables x et y de type *arbrelié* est souvent nécessaire :

- à chaque appel de la

procédure ajouter (in n : entier ; in x : arbrelié ; inout f : forêt) ;

il faut comparer *al* avec les éléments déjà présents dans *f* ;

- lorsqu'un substituant est construit, avant de le ranger dans *enssubst*, il faut s'assurer qu'il n'y est pas déjà ;

- si x et y sont des variables de type *arbrelié*, telles que leurs composantes *ar* ont la même complexité, il faut alors comparer leurs sous-*arbreliés*, c'est-à-dire les éléments de $x.ar.f$ et $y.ar.f$.

Ce dernier point implique que dans certains cas il faut parcourir tout un chemin dans les arborescences associées à chaque variable, et que par conséquent la comparaison est parfois coûteuse.

Exemple :

Considérons les deux substituants suivants :

/ CH2 / CH2 / CH2 'CL' et / CH2 / CH2 / CH2 'BR'

L'ordre des atomes est par exemple :

C < H < 'BR' < 'CL'

La comparaison de ces deux substituants entraîne la comparaison de :

- / CH2 / CH2 'CL' et / CH2 / CH2 'BR'
- / CH2 'CL' et / CH2 'BR'
- 'CL' et 'BR'

Pour éviter ces comparaisons multiples et coûteuses, il suffit de définir une fonction d'ordre O sur les éléments de *enssubst* et à valeurs dans un intervalle d'entiers, $[min, max]$, telle que :

$\forall x$ et y de type *arbrelié*, $x \leqslant y \iff O(x) \leqslant O(y)$

Cette fonction O doit également être définie pour le substituant fictif référencé par 0 dans les variables de type *forêt*. Ce substituant, que nous appelons s_o, est plus petit que tous les autres. Nous pouvons donc dès maintenant fixer :

$$O(s_o) = min$$

Cette fonction d'ordre est construite dynamiquement au fur et à mesure du remplissage du tableau *enssubst*, c'est-à-dire dans la procédure *rangersubst*. Le schéma de cette procédure est donc :

```
procédure rangersubst (in x : arbrelié ; inout lastsubst : indxsubst ;
                                          out index : indxsubst) ;
début
  'chercher x dans le tableau ensubst' ;
  si non trouvé
    alors
      'ajouter x dans enssubst et calculer la nouvelle fonction O' ;
  fsi ;
fin ; % rangersubst %
```

2.2. Construction dynamique d'une fonction d'ordre sur les objets de type *arbrelié*.

Soit *last* l'indice du dernier élément de *enssubst*.

Le domaine de définition de la fonction O est :

$$S(last) = \{s_o\} \cup \{enssubst[i],\ i \in [1,\ last]\}$$

Notre première idée, que nous avons programmée, a été d'utiliser pour O une bijection entre $S(last)$ et $[0,\ last]$. Cependant cette solution entraîne des mises à jour importantes. C'est pourquoi nous proposons une deuxième solution qui semble plus efficace, et dans laquelle O est définie seulement comme une injection de $S(last)$ dans $[min,\ max]$.

2.2.1. Utilisation d'une fonction d'ordre bijective.

La fonction d'ordre O est définie comme une bijection entre $S(last)$ et $[0,\ last]$. Quelle que soit la valeur de *last*, nous devons avoir :

$$\mathcal{O}(s_o) = 0$$

$$\forall n \in [1,\ last],\ \exists i \in [1,\ last], \mathcal{O}(enssubst[i]) = n$$

Notons $\mathcal{O}^{-1}$ la fonction réciproque de $\mathcal{O}$; alors

$$\forall n \in [0,\ last-1], \mathcal{O}^{-1}(n) < \mathcal{O}^{-1}(n+1)$$

Ces relations vont nous servir à calculer la fonction d'ordre au fur et à mesure qu'on ajoute un nouvel élément dans *enssubst*.

2.2.1.1. Calcul de la fonction d'ordre bijective.

Au départ le tableau *enssubst* est vide et $last = 0$. La fonction $\mathcal{O}$ est définie uniquement pour s_o et nous avons :

$$\mathcal{O}(s_o) = 0$$

Considérons maintenant que *enssubst* contient *last* éléments, que l'on connaît $\mathcal{O}(enssubst[i])$ pour chaque i de 1 à *last*, et que l'on ajoute un nouvel élément x à *enssubst* (x est rangé à l'indice $last+1$ et est distinct de $enssubst[i]$, pour tout i de 1 à *last*).

Notons $\underline{\mathcal{O}}$ la nouvelle fonction d'ordre définie sur $S(last) \cup \{x\}$ à valeurs dans $[0,\ last+1]$. Soit y le plus grand élément de $S(last)$ inférieur à x (y existe puisque $S(last)$ contient toujours s_o qui est le plus petit des éléments de $S(last)$). Il est clair que nous avons :

$$\forall z \in S(last),\ (z \leqslant y \Longrightarrow \underline{\mathcal{O}}(z) = \mathcal{O}(z))$$
$$\text{et } (z > y) \Longrightarrow \underline{\mathcal{O}}(z) = \mathcal{O}(z) + 1$$
$$\underline{\mathcal{O}}(x) = \mathcal{O}(y) + 1$$

Pour calculer la fonction $\underline{\mathcal{O}}$ il faut chercher d'abord l'élément $y = \max\{z \in S(last),\ z < x\}$. Pour cela on peut chaîner tous les éléments de $S(last)$ dans l'ordre croissant, par exemple. Mais nous pouvons également utiliser la fonction $\mathcal{O}^{-1}$, ce qui a l'avantage de permettre une recherche dichotomique de y. En effet nous avons :

$$\forall n \in [0,\ last-1], \mathcal{O}^{-1}(n) < \mathcal{O}^{-1}(n+1)$$

Posons $n = \mathcal{O}(y)$ et $k = \mathcal{O}(z)$. Alors :

$$\forall z \in S(last),\ z \leqslant y \Longrightarrow \underline{\mathcal{O}}(z) = \mathcal{O}(z)$$
$$\Longleftrightarrow \forall k \in [0,\ n],\ \underline{\mathcal{O}}(\mathcal{O}^{-1}(k)) = \mathcal{O}(\mathcal{O}^{-1}(k)) = k$$
$$\Longleftrightarrow \forall k \in [0,\ n],\ \underline{\mathcal{O}}^{-1}(k) = \mathcal{O}^{-1}(k)$$

$$\forall z \in S(last),\ z > y \Longrightarrow \underline{\mathcal{O}}(z) = \mathcal{O}(z) + 1$$
$$\Longleftrightarrow \forall k \in [n+1,\ last],\ \underline{\mathcal{O}}(\mathcal{O}^{-1}(k)) = \mathcal{O}(\mathcal{O}^{-1}(k)) + 1 = k+1$$
$$\Longleftrightarrow \forall k \in [n+1,\ last],\ \underline{\mathcal{O}}^{-1}(k+1) = \mathcal{O}^{-1}(k)$$

$$\underline{\mathcal{O}}(x) = \mathcal{O}(y) + 1$$
$$\Longleftrightarrow \underline{\mathcal{O}}(x) = \mathcal{O}(\mathcal{O}^{-1}(n)) + 1 = n+1$$
$$\Longleftrightarrow \underline{\mathcal{O}}^{-1}(n+1) = x$$

Les nouvelles fonctions $\underline{\mathcal{O}}$ et $\underline{\mathcal{O}}^{-1}$ sont donc définies par :

$$\forall k \in [0,\ n],\ \underline{\mathcal{O}}(\mathcal{O}^{-1}(k)) = k$$
$$\underline{\mathcal{O}}^{-1}(k) = \mathcal{O}^{-1}(k)$$
$$\forall k \in [n+1,\ last],\ \underline{\mathcal{O}}(\mathcal{O}^{-1}(k)) = k+1$$
$$\underline{\mathcal{O}}^{-1}(k+1) = \mathcal{O}^{-1}(k)$$
$$\underline{\mathcal{O}}(x) = n+1$$
$$\underline{\mathcal{O}}^{-1}(n+1) = x$$

où $k = \mathcal{O}(\max\{z \in S(last),\ z < x\})$

Cherchons maintenant l'algorithme de la procédure *rangersubst* utilisant cette fonction d'ordre bijective.

2.2.1.2. Procédure *rangersubst* définie pour une fonction d'ordre bijective.

Les valeurs des fonctions $\mathcal{O}$ et $\mathcal{O}^{-1}$ pour les éléments de $S(last)$ sont conservées dans un tableau de la forme :

```
var ordre : tableau [indxsubst] de
                struct
                  valeur,
                  inverse : indxsubst ;
                fin ;
```

Le premier élément $ordre[o]$ correspond à s_o et est initialisé par :

```
ordre[0].valeur := 0 ;
ordre[0].inverse  := 0 ;
```

Pour chaque i compris entre 1 et *last*, $ordre[i].valeur$ contient la valeur $\mathcal{O}(enssubst[i])$, et $ordre[i].inverse$ contient l'indice de l'élément x tel que $\mathcal{O}(x) = i$.

Exemple :

Considérons le tableau *enssubst* qui contient les substituants s_c, s_t, s_r, s_l, s_i, s_g, s_w, s_p. Pour des raisons de commodités, nous avons donné des indices alphabétiques à ces substituants, et nous supposerons qu'ils sont classés selon l'ordre de ces indices. Nous donnons ci-dessous le tableau *enssubst* et le tableau *ordre*, pour ces constituants.

1	s_c
2	s_t
3	s_r
4	s_l
5	s_i
6	s_g
7	s_w
lastsubst ← 8	s_p

tableau *enssubst*

	valeur	inverse
0	0	0
1	1	1
2	7	6
3	6	5
4	4	4
5	3	8
6	2	3
7	8	2
8	5	7

tableau *ordre*

Lorsque l'on doit ranger un nouvel élément x dans le tableau, il faut d'abord s'assurer qu'il n'y est pas déjà. Ceci est fait par une procédure *chercher* dont l'interface est :

```
procédure chercher (in x : arbrelié ; in last : indxsubst ;
                    out trouvé : booléen ; out n : indxsubst) ;
```

last est le dernier élément de *enssubst*.

Le résultat de cette procédure est défini par le booléen *trouvé* et un indice *n* de *indxsubst* :

si *trouvé* est *vrai*, alors *n* est l'indice de x dans *enssubst*, sinon *n* contient la valeur de *ordre*[*l*].*valeur* où *l* est l'indice du plus grand élément de *enssubst* inférieur à x.

Comme nous l'avons indiqué, nous pouvons faire une recherche dichotomique de x en utilisant le champ *inverse* du tableau *ordre*. Il est nécessaire de comparer deux variables de type *arbrelié* ; nous introduisons pour cela le

```
type relation = (inférieur, égal, supérieur) ;
```

et la

```
fonction cmpsubst (x, y : arbrelié) : relation ;
```

L'algorithme de cette fonction est donnée au paragraphe 2.3 et est tel que :

cmpsubst (*x*, *y*) = *inférieur* ⟺ $x < y$
cmpsubst (*x*, *y*) = *égal* ⟺ $x = y$
cmpsubst (*x*, *y*) = *supérieur* ⟺ $x > y$

```
procédure chercher (in x : arbrelié ; in last : indxsubst ;
                    out trouvé : booléen ; out n : indxsubst) ;
var i, k : indxsubst ;
début
  i := 1 ; n := last ;            % initialisations %
  trouvé := faux ;
  tantque non trouvé et i ⩽ n faire
    k := (i + n) div 2 ;
      % comparaison de x avec l'élément d'indice ordre[k].inverse
      et branchement en fonction du résultat %
    cas cmpsubst (x, enssubst [ordre[k].inverse]) de
      inférieur : n := k - 1 ;
      égal      : début
                    trouvé := vrai ;
                    n := ordre[k].inverse ;
                  fin ;
      supérieur : i := k + 1 ;
    fcas ;
  ftantque ;
    % si trouvé est faux, n est la valeur cherchée %
fin ; % chercher %
```

Si la procédure *chercher* ne trouve pas x dans *enssubst*, il faut le ranger à l'indice *last + 1* (après avoir vérifier que *last < maxsubst*), et mettre à jour le tableau *ordre*. Cette mise à jour est faite par la

procédure màjordre (in n, last : indxsubst) ;

où *last* est l'indice du dernier élément de *enssubst* avant l'ajout de x et où n est l'entier qui définit le classement du nouvel élément x ; n est tel que : ($n = 0$ et $s_o < x <$ *enssubst [ordre[1].inverse]*) ou ($n > 0$ et *enssubst [ordre[n].inverse]* $< x <$ *enssubst [ordre[n + 1].inverse]*).

Les relations qui définissent les fonctions $\underline{O}$ et $\underline{O}^{-1}$ montrent que les valeurs modifiées dans *ordre* sont *ordre[ordre[k].inverse].valeur* et *ordre[k + 1].inverse*, pour k dans [*n + 1, last*]. D'autre part la nouvelle valeur de *ordre[k + 1].inverse*, est celle qui est contenue dans *ordre[k].inverse*. Il faut donc faire la mise à jour dans l'ordre des indices k décroissants.

```
procédure màjordre (in n, last : indxsubst) ;
var k : indxsubst ;
début
  pour k := last decr n + 1 faire
    ordre[ordre[k].inverse].valeur := k + 1 ;
    ordre[k + 1].inverse := ordre[k].inverse ;
  fpour ;
  ordre[last + 1].valeur := n + 1 ;
  ordre[n + 1].inverse := last + 1 ;
fin ; % màjordre %
```

Remarque :

Si l'élément x est plus petit que tous ceux de *enssubst*, c'est-à-dire si $n = 0$, tous les éléments de *ordre*, sauf *ordre[0]* sont modifiés.

Exemple :

Nous donnons les modifications des tableaux *enssubst* et *ordre* qu'entraîne l'ajout de s_h à l'ensemble $\{s_c, s_t, s_r, s_l, s_i, s_g, s_w, s_p\}$. s_h se place après s_g ; le résultat de la procédure *chercher* est :

trouvé = faux et $n = 2$ (valeur de $O(s_g)$)

1	s_c	
2	s_t	
3	s_r	
4	s_l	
5	s_i	
6	s_g	
7	s_w	
lastsubst ← 8	s_p	
lastsubst ← 9	s_h	

tableau *enssubst*

	valeur	inverse
0	0	0
1	1	1
2	~~7~~ 8	6
3	~~6~~ 7	~~5~~ 9
4	~~4~~ 5	~~4~~ 5
5	~~3~~ 4	~~8~~ 4
6	2	~~3~~ 8
7	~~8~~ 9	~~2~~ 3
8	~~5~~ 6	~~7~~ 2
9	3	8

tableau *ordre*

Avec ces deux procédures, *chercher* et *màjordre*, l'algorithme de *rangersubst* est le suivant :

```
procédure rangersubst (in x : arbrelié ; inout lastsubst : indxsubst ;
                       out index : indxsubst) ;
var trouvé : booléen ;
début
  chercher (x, lastsubst, trouvé, index) ;
  si trouvé
    alors ; % enssubst et ordre restent inchangés.
              index est l'indice de x dans enssubst %
    sinon
      si last ≥ maxsubst
```

```
            alors
              'signaler une erreur : débordement de enssubst'
            sinon
                        % rangement de x à la fin de enssubst %
              lastsubst := lastsubst + 1 ;
              enssubst [lastsubst] := x ;
                        % mise à jour du tableau ordre %
              màjordre (index, lastsubst - 1) ;
              index := lastsubst ;
          fsi ;
        fsi ;
      fin ; % rangersubst %
```

Comme nous l'avons indiqué, la définition d'une fonction d'ordre bijective est coûteuse en temps, car les mises à jour du tableau *ordre* sont parfois importantes. Nous proposons donc une autre solution en définissant une fonction d'ordre injective, et nous donnerons ensuite une version de la procédure *rangersubst* associée à cette fonction.

2.2.2. Utilisation d'une fonction d'ordre injective.

2.2.2.1. Calcul de la fonction d'ordre injective.

Comme dans le cas bijectif, la fonction d'ordre injective est définie au fur et à mesure du rangement d'un nouvel élément dans *enssubst*. Son domaine de définition est

$$S(last) = \{s_o\} \cup \{enssubst[i],\ i \in [1,\ last]\}$$

où *last* est l'indice du dernier élément de *enssubst*, mais elle prend ses valeurs dans un intervalle [*min*, *max*] plus grand que [*0*, *last*]. Cet intervalle est constant et reste à définir. s_o étant toujours le plus petit élément de $S(last)$, nous fixons :

$$O(s_o) = min$$

La construction progressive de O est faite de telle sorte que l'ajout d'un nouvel élément à *enssubst* ne modifie pas, dans la mesure du possible, les valeurs de O présentes dans *enssubst*. Les premiers éléments de *enssubst* sont construits au cours de la lecture des réactifs d'un mécanisme. La spécification des fonctions sémantiques associées à la grammaire d'un constituant entraîne

que, en général, les premiers substituants construits sont les plus petits, et les derniers sont les plus grands.

Exemple :

Lors de la lecture de la notation suivante :

C(//O)H / C(C(//O)/H)(C(CH2/H)(CH3) / CH3) / C(///C/O/H)

les substituants construits sont, dans l'ordre :

s_1 = // O
s_2 = / H
s_3 = / CH // O
s_4 = / CH3
s_5 = / C(CH3)3
s_6 = / OH
s_7 = /// C / OH
s_8 = / C /// C / OH
s_9 = / C(CH // O)(C(CH3)3) / C /// C / OH

Du fait de la cohérence de l'ordre, nous avons :

$s_1 < s_3 < s_9$
$s_2 < s_3 < s_9$
$s_1 < s_4 < s_5 < s_9$
$s_1 < s_6 < s_7 < s_8 < s_9$

Suite à cette remarque, pour calculer la fonction O, nous introduisons une variable entière globale *incr*. Nous fixerons sa valeur plus tard. La valeur de O(x) pour un nouvel élément est alors définie par :

- si y désigne le plus grand élément de $S(last)$ et si x est plus grand que y alors $O(x) = O(y) + incr$;

- si x est classé entre les deux éléments y et z de $S(last)$ $(y < z)$ alors $O(x) = \frac{O(y) + O(z)}{2}$ (division entière).

Dans le premier cas il faut que $O(y) + incr$ soit inférieur ou égal à *max* ; dans le second cas il faut que $\frac{O(y) + O(z)}{2}$ soit différent de $O(y)$.

Pour déterminer les valeurs de *min*, *max* et *incr*, il faut remarquer que le calcul de O revient à étalonner l'intervalle [*min*, *max*] par des valeurs

*min + k * incr*. Puis les sous-intervalles [*min + k * incr, min + (k + 1) × incr*], de longueur *incr*, sont partagés en deux, en quatre, en huit, etc... Il est donc intéressant de prendre pour *incr* une puissance de 2.

D'autre part nous avons fixé à 200 le nombre d'éléments de *enssubst*. Alors la valeur maximum prise par O est $min + 200 * incr$. Si nous choisissons *max* supérieur ou égal à cette valeur, nous nous assurons que toutes les valeurs prises par O sont inférieures ou égales à *max*. A titre indicatif, nous pouvons fixer :

$$min = -2^{15}$$
$$max = 2^{15} - 1 \ .$$
$$incr = 2^{8}$$

Remarque

Le fait de choisir $[min, max] = [-2^{15}, 2^{15} - 1]$ permet de stocker les valeurs de O sur 16 bits.

Avec ces valeurs, une modification de la fonction O est nécessaire uniquement si x est intercalé entre deux substituants y et z tels que :

$$O(z) = O(y) + 1$$

La valeur de $O(x)$, calculée par $\frac{O(y) + O(z)}{2}$ est alors égale à $O(y)$. Dans ce cas on recalcule toutes les valeurs de O par l'expression $min + k * incr$.

Dans cette mise à jour la valeur de *incr* reste constante, ce qui est possible car *maxsubst* est fixée à 200. Mais pour une constante *maxsubst* plus grande, le choix de *min*, *max* et *incr* vérifiant la relation

$$min + maxsubst * incr \leqslant max$$

peut entraîner une valeur de *incr* trop faible. Dans ce cas il est peut-être nécessaire de choisir au départ une valeur plus grande pour *incr*, quitte à vérifier que les valeurs de O sont inférieures à *max*, et à réajuster cette valeur à chaque mise à jour. Pour prévoir cette transformation, nous définissons *incr* comme une variable globale qu'il faut initialiser en début de programme, et non comme une constante.

Exemple :

Reprenons les substituants de l'exemple précédent. En prenant $H < 0$, ils sont classés dans l'ordre suivant :

$$s_2 < s_1 < s_6 < s_7 < s_8 < s_4 < s_3 < s_5 < s_9$$

En effet

- s_1, s_2, s_6, s_7 et s_8 sont de complexité 1
- s_3 et s_4 sont de complexité 2
- s_5 et s_9 sont de complexité 3

D'autre part :

$$H < 0 \implies s_2 < s_1 \implies s_4 < s_3$$

Le calcul de la fonction d'ordre fournit les valeurs suivantes :

$$O(s_1) = O(s_o) + incr = min + incr = -2^{15} + 256$$

$$O(s_2) = \frac{O(s_o) + O(s_1)}{2} = min + \frac{incr}{2} = -2^{15} + 128$$

$$O(s_3) = O(s_1) + incr = min + 2\ incr = -2^{15} + 512$$

$$O(s_4) = \frac{O(s_1) + O(s_3)}{2} = min + incr + \frac{incr}{2} = -2^{15} + 384$$

$$O(s_5) = O(s_3) + incr = min + 3\ incr = -2^{15} + 768$$

$$O(s_6) = \frac{O(s_1) + O(s_4)}{2} = min + incr + \frac{incr}{4} = -2^{15} + 320$$

$$O(s_7) = \frac{O(s_6) + O(s_4)}{2} = min + incr + \frac{incr}{4} + \frac{incr}{8} = -2^{15} + 328$$

$$O(s_8) = \frac{O(s_7) + O(s_4)}{2} = min + incr + \frac{incr}{4} + \frac{incr}{8} + \frac{incr}{16} = -2^{15} + 332$$

$$O(s_9) = O(s_5) + incr = min + 4\ incr = -2^{15} + 1024$$

Cette nouvelle définition de la fonction d'ordre entraîne bien sûr la modification de la procédure *rangersubst*.

2.2.2.2. Procédure *rangersubst* définie pour une fonction d'ordre injective.

Comme O est une fonction injective, il n'est pas question de définir sa fonction *inverse*. En outre la recherche d'un élément x dans $S(last)$ doit fournir, en cas d'échec, les deux éléments successifs y et z de $S(last)$ tels que

$$y < x < z$$

Pour faire cette recherche, une liste chaînée des éléments de *S(last)* dans l'ordre croissant peut convenir. Mais on peut aussi construire un arbre binaire, tel que pour tout sous-arbre de racine x, tous les éléments du sous-arbre gauche sont inférieurs à x, tous ceux du sous-arbre droit sont supérieurs à x. Cette solution est plus lourde à mettre en oeuvre, mais elle active la recherche d'un élément.

Pour implanter cette solution, il faut définir le tableau *ordre* comme suit :

```
var ordre : tableau [0..maxsubst] de
            struct
              valeur : min..max ;
              gauche,
              droite : indxsubst ;
            fin ;
```

Au départ tous les éléments du tableau sont initialisés à 0, sauf *ordre*[*o*].*valeur* qui est initialisée à *min*.

Soit i un indice compris entre 0 et *lastsubst* ;

- dans *enssubst*[i].*gauche*, est rangé l'indice du premier substituant construit après *enssubst*[i] et inférieur à celui-ci ;

- dans *enssubst*[i].*droite*, est rangé l'indice du premier substituant construit après *enssubst*[i] et supérieur à celui-ci.

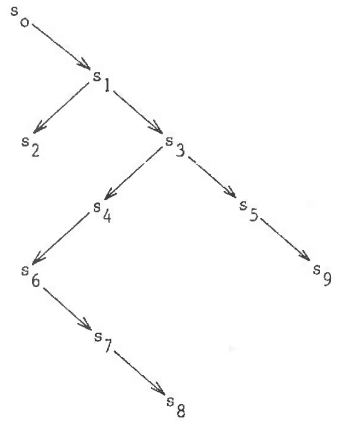
Exemple :

Pour les substituants s_i, i de 1 à 9, de l'exemple précédent, les tableaux *enssubst* et *ordre* sont les suivants :

1	s_1
2	s_2
3	s_3
4	s_4
5	s_5
6	s_6
7	s_7
8	s_8
9	s_9

	valeur	*gauche*	*droite*
0	$0(s_0)$ = *min*	0	1
1	$0(s_1)$	2	3
2	$0(s_2)$	0	0
3	$0(s_3)$	4	5
4	$0(s_4)$	6	0
5	$0(s_5)$	0	9
6	$0(s_6)$	0	7
7	$0(s_7)$	0	8
8	$0(s_8)$	0	0
9	$0(s_9)$	0	0

Le tableau *ordre* est une représentation de l'arbre binaire suivant :



La spécification du tableau *ordre* implique que la racine de l'arbre binaire qu'il représente est s_o. Mais ce substituant est fictif et n'existe pas dans *enssubst*. D'autre part cet élément étant plus petit que tous ceux de *enssubst*, il n'a jamais de sous-arbre gauche. En conséquence, on fera la recherche d'un élément dans le sous-arbre droit de s_o, dont la racine est toujours s_1. Cependant l'existence de l'élément *ordre[0]* permet de stocker dans son champ *valeur* la valeur de *min*, et ainsi de faire rentrer dans le cas général le calcul de l'ordre pour un substituant inférieur à tous ceux déjà présents dans *enssubst*, ce qui élimine le test correspondant.

L'interface de la procédure *chercher* est la suivante :

```
procédure chercher (in x : arbrelié ;
                    out trouvé : booléen ; out inf, sup : indxsubst) ;
```

- si *trouvé* est *vrai*, x est contenu dans *enssubst* à l'indice *inf* ;
- si *trouvé* est *faux*, x n'est pas dans *enssubst* ; le classement de x par rapport aux éléments de *enssubst* est alors défini par les valeurs de *inf* et *sup* comme suit :

- si *inf* est nul, x est plus petit que tous les éléments de *enssubst*, sinon *inf* est l'indice du plus grand élément inférieur à x.

- si *sup* est nul, x est plus grand que tous les éléments de *enssubst*, sinon *sup* désigne l'indice du plus petit élément supérieur à x.

```
procédure chercher (in x : arbrelié ;
                    out trouvé : booléen ; out inf, sup : indxsubst) ;
var i : indxsubst ;
début
  i := ordre[0].droite ;              % initialisations %
  inf := 0 ; sup := 0 ;
  trouvé := faux ;
  tantque non trouvé et i ≠ 0 faire
    cas cmpsubst (x, enssubst[i]) de
      inférieur : début
                    sup := i ; i := ordre[i].gauche ;
                  fin ;
      égal      : début
                    trouvé := vrai ; inf := i ;
                  fin ;
```



```
      supérieur : début
                    inf := i ; i := ordre[i].droite ;
                  fin ;
    fcas ;
  ftantque ;
fin ; % chercher %
```

Remarque :

On initialise *i* avec la valeur de *ordre[0].droite*, et non avec *1*, pour traiter le cas où *enssubst* est vide.

Si l'élément x n'est pas dans le tableau, il faut le ranger à la fin de *enssubst*, et mettre à jour le tableau *ordre*, c'est-à-dire calculer *O(x)* et ajouter x dans l'arbre binaire. D'après la définition de *inf* et *sup* nous avons :

- si *inf* < *sup*, x est une feuille gauche de l'élément d'indice *sup* ;
- si *inf* > *sup*, x est une feuille droite de l'élément d'indice *inf*.

L'égalité entre *inf* et *sup* n'est rencontrée que lorsque *inf* et *sup* sont nuls ; x est alors le premier élément rangé dans *enssubst* et c'est la feuille droite de s_o. L'interface nécessaire à la procédure *màjordre* est donc :

```
procédure màjordre (in ix, inf, sup : indxsubst) ;
```

ix est l'indice du nouvel élément x.

La modification de la fonction *ordre*, nécessaire lorsque *sup* est non nul et *ordre[inf].valeur* est égal à *ordre[sup].valeur + 1*, nécessite un parcours de l'arbre du type "sous-arbre gauche - racine - sous-arbre droit" pour rencontrer les valeurs dans l'ordre croissant.

Ce parcours est fait par la

```
procédure parcours (in i : indxsubst) ;
```

qui est récursive et interne à *màjordre*.

Ecrivons les algorithmes de *màjordre* et *parcours* :

```
procédure màjordre (in ix, inf, sup : indxsubst) ;
var k : min..max ;
```

```
procédure parcours (in i : indxsubst) ;

début
  si i = 0
    alors ; % l'arbre est vide %
    sinon
      parcours (ordre[i].gauche) ;
      k := k + incr ;
      ordre[i].valeur := k ;
      parcours (ordre[i].droite) ;
  fsi ;
fin ;  % parcours %

début
                              % ajout de enssubst[ix] dans l'arbre %
  si inf ⩾ sup
    alors ordre[inf].droite := ix ;
    sinon ordre[sup].gauche := ix ;
  fsi ;
                              % calcul de ordre[ix].valeur %
  si sup = 0
    alors ordre[ix].valeur := ordre[inf].valeur + incr ;
    sinon
      si ordre[sup].valeur = ordre[inf].valeur + 1
        alors
          k := min ; parcours (1) ;
        sinon ordre[x].valeur := (ordre[inf].valeur
                                  + ordre[sup].valeur) div 2 ;
      fsi ;
  fsi ;
fin ; % majordre %
```

Remarque :

Vu l'ordre de grandeur des valeurs données à *maxsubst*, *min*, *max* et *incr*, la procédure *parcours* est appelée peu souvent.

Avec ces deux procédures, l'algorithme de *rangersubst* s'écrit :


```
procédure rangersubst (in x : arbrelié ; inout lastsubst : indxsubst ;
                       out index : indxsubst) ;
var trouvé : booléen ;
    sup : indxsubst ;

début
  chercher (x, trouvé, index, sup) ;
  si trouvé
    alors ; % x est dans enssubst à l'indice index %
    sinon
      si lastsubst ⩾ maxsubst
        alors 'signaler erreur : débordement de enssubst' ;
        sinon
                              % rangement de x dans enssubst %
          lastsubst := lastsubst + 1 ;
          enssubst[lastsubst] := x ;
                              % mise à jour de ordre %
          majordre (lastsubst, index, sup) ;
          index := lastsubst ;
      fsi ;
  fsi ;
fin ; % rangersubst %
```

Pour terminer ce paragraphe sur la relation d'ordre définie sur les substituants, il reste à écrire l'algorithme de la comparaison de deux substituants.

2.3. Comparaison de deux objets de type *arbrelié*.

Nous avons vu que l'ordre défini est identique à l'ordre de type lexicographique défini sur l'ensemble des suites :

$\{(x.ar.cplx,\ x.ar.f,\ x.ar.at,\ x.l),\ x$ *de type arbrelié*$\}$.

D'après la spécification donnée pour le type *forêt*, une variable f de ce type est telle que :

- les éléments $f.descr[i]$, pour i de 1 à $f.nc$ sont classés dans l'ordre décroissant ;

- les éléments $f.descr[i]$, pour i de $f.nc+1$ à $maxval$ sont vides, c'est-à-dire que les champs *repr* et *nb* ont la valeur zéro.

En associant à zéro le substituant fictif s_o plus petit que tous les autres, et d'après l'ordre défini sur les éléments de type *jumeaux*, il est équivalent de considérer les suites de 11 éléments de la forme :

(x.ar.cplx,
fx.descr[1].repr,
fx.descr[1].nb,
fx.descr[2].repr,
fx.descr[2].nb,
fx.descr[3].repr,
fx.descr[3].nb,
fx.descr[4].repr,
fx.descr[4].nb,
x.ar.at
x.ar.l) ;

où x est de type *arbrelié* et $fx = x.ar.f$.

Cependant les éléments d'une telle suite sont de types différents et chaque type a son ordre propre :

- $x.ar.cplx$ et $fx.descr[i].nb$, i de *1* à *maxval*, sont de type *entier*, et l'ordre est celui de $\mathbb{N}$;

- $x.ar.at$ est de type *entier*, et l'ordre est celui des *atomes* ;

- $fx.descr[i].repr$, i de *1* à *maxval*, sont de type *arbrelié*, et l'ordre est celui des objets *arbrelié*.

- $x.l$ est de type *liaison* et l'ordre est celui des liaisons.

Une comparaison directe de ces éléments exige, en PASCAL, une procédure distincte pour chaque type. Pour supprimer cet inconvénient, il suffit de disposer pour chaque type T d'une fonction d'ordre f_T définie sur les objets de type T et à valeurs dans $\mathbb{N}$. Ces fonctions doivent évidemment vérifier :

$$\forall x_1,\ x_2 \text{ de type } T, \qquad x_1 \leqslant x_2 \iff f_T(x_1) \leqslant f_T(x_2)$$

A partir d'une variable x de type *arbrelié*, on construit un vecteur vx dont le ième élément, de type T, est l'image par f_T du ième élément de la suite décrite ci-dessus.

D'après ce qui précède nous utilisons

- pour f_{entier} et f_{atome} l'identité sur $\mathbb{N}$;

- pour $f_{liaison}$, la

fonction rang (l : liaison) : entier ;

utilisée pour le calcul de la valence d'un atome (cf. chap. III, § 3.2.4) ;

- pour $f_{arbrelié}$ la fonction définie par :

$f_{arbrelié}\ (so) = ordre[0].valeur$
$\forall i \in [1,\ lastsubst],\ f_{arbrelié}(enssubst[i]) = ordre[i].valeur$

Remarque

f_{atome} est l'identité, car, dans un premier prototype, nous avons choisi sur les atomes l'ordre induit par le code lexicographique, qui est utilisé pour identifier les atomes dans les variables de type *arbre*.

Avec ces fonctions la forme du vecteur vx associé à x est $(fx = x.ar.f)$:

$vx[0] = x.ar.cplx$
$vx[1] = ordre[fx.descr[1].repr].valeur$
$vx[2] = fx.descr[1].nb$
$vx[3] = ordre[fx.descr[2].repr].valeur$
$vx[4] = fx.descr[2].nb$
$vx[5] = ordre[fx.descr[3].repr].valeur$
$vx[6] = fx.descr[3].nb$
$vx[7] = ordre[fx.descr[4].repr].valeur$
$vx[8] = fx.descr[4].nb$
$vx[9] = x.ar.at$
$vx[10] = rang(x.l)$

La comparaison de deux variables x et y se fait donc en

- construisant les vecteurs vx et vy associés respectivement à x et y ;
- comparant lexicographiquement ces deux vecteurs dont tous les éléments sont de type *entier*.

Nous définissons le

```
type vecteur = tableau [0..10] de entier ;
```

et la

```
procédure constvect (in x : arbrelié ; out vx : vecteur) ;
```

Ecrivons les deux procédures *constvect* et *cmpsubst*.

```
procédure constvect (in x : arbrelié ; out vx : vect) ;

var i : entier ;

début
  avec x.ar faire
    v[0] := cplx ;
    pour i := 1 à maxval faire
      avec f.descr[i] faire
        vx[2 *i - 1] := ordre[repr].valeur ;
        vx[2 *i] := nb ;
      favec ;
    fpour ;
    v[9] := at ;
  favec ;
    v[10] := x.l ;
fin ; % constvect %

fonction  cmpsubst(x,y : arbrelié) : relation ;

% x < y ⟹ cmpsubst(x,y) = inférieur ;
  x = y ⟹ cmpsubst(x,y) = égal ;
  x > y ⟹ cmpsubst(x,y) = supérieur ; %

var vx, vy : vecteur ;
    i : entier ;

début
  constvect (x,vx) ; constvect (y,vy) ;
  i := 0 ;
  tantque i < 10 et vx[i] = vy[i] faire
    i := i + 1 ;
  ftantque ;
```



```
  si vx[i] = vy[i]
    alors cmpsubst := égal ;
    sinon si vx[i] < vy[i]
            alors cmpsubst := inférieur ;
            sinon cmpsubst := supérieur ;
          fsi ;
  fsi ;
fin ; % cmpsubst %
```

L'implantation de la relation d'ordre (totale) sur les objets de type *arbrelié* entraîne une gestion assez lourde du tableau *enssubst* au niveau de la place mémoire. Mais si l'on ne prend pas la précaution de classer tous les substituants au fur et à mesure de leur construction, et de conserver ce classement dans le tableau *ordre*, la comparaison de deux substituants est beaucoup plus longue. D'autre part la

```
procédure ajouter (in n : entier ; in x : arbrelié ; inout f : forêt) ;
```

est essentielle dans la construction des constituants lors de la lecture des réactifs, de la recherche du foyer dans une molécule et de la création des processus élémentaires. Or il faut ajouter x dans f en conservant l'ordre décroissant de ses éléments. Au lieu de passer le paramètre x de type *arbrelié*, nous passons son indice ix dans *enssubst*. L'interface de la procédure devient :

```
procédure ajouter (in n : entier ; in ix : indxsubst ; inout f : forêt) ;
```

Lorsque x est un substituant nouvellement construit, il faut appeler la procédure *rangersubst* avant la procédure *ajouter* pour connaître son indice dans *enssubst*. Avec cette nouvelle interface, les comparaisons nécessaires à l'ajout de *enssubst*[ix] dans f sont effectuées sur des entiers par l'intermédiaire du tableau *ordre*, et non plus sur des variables de type *arbrelié*.

Le temps perdu à gérer le tableau *ordre* est donc largement gagné par les exécutions de la procédure *ajouter*. De plus ce tableau permet de faire une recherche dichotomique dans *enssubst*.

3. LECTURE DES REACTIFS D'UN MECANISME REACTIONNEL.

La création d'un mécanisme réactionnel débute par la lecture des réactifs qui sont tous des constituants moléculaires. Ces réactifs sont en nombre quelconque et se présentent sous forme d'une liste de molécules séparées par une virgule et terminée par un point virgule. La syntaxe de cette liste de réactifs est définie par la grammaire suivante, dans laquelle nous avons repris la grammaire des molécules du chapitre III.

3.1. Grammaire d'une liste de réactifs.

a) Vocabulaire terminal.

{atome, liaison, entier, po, pf, deux, virg, ptvirg}

où :

virg ::= ,
ptvirg ::= ;

Les autres mots terminaux sont définis comme au chapitre III.

b) Vocabulaire non terminal.

{<réactifs>, <molécule>, <molsym>, <groupe>, <listesubst>, <subst>, <L>, <M>}

c) Axiome.

<réactifs>

d) Règles de production.

```
<réactifs> ::= <molécule> ptvirg
             | <molécule> virg <réactifs>
<molécule> ::= <molsym> | <groupe>
  <molsym> ::= <L> atome deux
             | <L> po <groupe> pf deux
  <groupe> ::= atome <listesubst>
<listesubst> ::= <subst> <M> <listesubst>
             | liaison <groupe>
             | Λ
```



```
<subst> ::= atome
          | po <L> <groupe> pf
    <L> ::= liaison | Λ
    <M> ::= entier | Λ
```

Pour implanter cette grammaire, il est nécessaire de la mettre sous forme LL(1).

3.2. Grammaire LL(1) d'une liste de réactifs.

La transformation de la grammaire est faite selon la méthode développée dans [GRIFFITHS 1974], [FOSTER 1968], [KNUTH 1971b]. Elle implique l'introduction de trois nouveaux non terminaux <suiteréactifs>, <suitemol> et <suitemolsym>. Le non terminal <molsym> disparaît.

```
      <réactifs> ::= <molécule> <suiteréactifs>
<suiteréactifs> ::= virg <réactifs>
                  | ptvirg
      <molécule> ::= atome <suitemol>
                  | liaison <suitemolsym> deux
                  | po <groupe> pf deux
      <suitemol> ::= <listesubst>
                  | deux
<suitemolsym> ::= atome
                  | po <groupe> pf
       <groupe> ::= atome <listesubst>
  <listesubst> ::= <subst> <M> <listesubst>
                  | liaison <groupe>
                  | Λ
        <subst> ::= atome
                  | po <L> <groupe> pf
           <L> ::= liaison
                  | Λ
           <M> ::= entier
                  | Λ
```

La règle :

<molécule> ::= atome <suitemol>

produit soit une molécule sous forme symétrique, si <suitemol> produit le mot terminal deux, soit une molécule sous forme non symétrique si <suitemol> produit le non terminal <listesubst>.

Les autres règles dérivant de <molécule> correspondent à des formes symétriques.

3.3. Sémantique.

La sémantique de la grammaire doit construire les représentations internes des molécules, en vérifiant que la valence des atomes est correcte, et les stocker dans le tableau *ensmol*, sous leur représentation canonique. Le stockage est fait par la

procédure rangermol (in m : molécule ;
inout lastm : indxcnst ;
out index : indxcnst) ;

introduite au chapitre IV.

Nous avons vu que, si une molécule est écrite sous forme symétrique, la représentation interne construite est automatiquement canonique. La recherche d'une représentation canonique n'est donc nécessaire que pour une molécule écrite sous forme non symétrique ; elle est intégrée à la fonction sémantique *constmolnonsym*.

Les attributs des non terminaux <suitemol> et <suitemolsym> sont définis dans le tableau ci-dessous.

nom	nature	type	signification
at.<suitemol>	hérité	*atome*	code de l'atome lu par la règle <molécule> ::= atome <suitemol>
m.<suitemol>	synthétisé	*molécule*	molécule produite par <suitemol>
ar.<suitemolsym>	synthétisé	*arbre*	arbre produit par <suitemolsym>
v.<suitemolsym>	hérité	*entier*	contribution de la liaison de symétrie à la valence de l'atome racine de *ar.<suitemolsym>*

Les éléments d'une variable de type *forêt* sont les indices des substituants dans *enssubst*. En conséquence nous remplaçons l'attribut *al.<subst>* de type *arbrelié* par l'attribut *s.<subst>* de type *indxsubst*. Celui-ci est obtenu en appelant la procédure *rangersubst*.

Nous décrivons maintenant le calcul des attributs.

<réactifs> ::= <molécule> <suiteréactifs>
rangermol (m.<molécule>, lastm, index) ;

<réactifs> ::= virg <réactifs>

<réactifs> ::= ptvirg

<molécule> ::= atome <suitemol>
at.<suitemol> := at.atome ;
m.<molécule> := m.<suitemol> ;

<molécule> ::= liaison <suitemolsym> deux
v.<suitemolsym> := rang (l.liaison) ;
m.<molécule> := constmolsym (lier (l.liaison, ar.<suitemolsym>)) ;

<molécule> ::= po <groupe> pf deux
v.<groupe> := 1 ;
m.<molécule> := constmolsym (lier (simple, ar.<groupe>)) ;

<suitemol> ::= <listesubst>
stockerval (at.<suitemol>, v.<listesubst>) ;
m.<suitemol> := constmolnonsym (enraciner (at.<suitemol>, f.<listesubst>)) ;

<suitemol> ::= deux
stockerval (at.<suitemol>, 1) ;
m.<suitemol> := constmolsym (lier (simple, enraciner (at.atome, fvide))) ;

<suitemolsym> ::= atome
stockerval (at.atome, v.<suitemolsym>) ;
ar.<suitemolsym> := enraciner (at.atome, fvide) ;

<suitemolsym> ::= po <groupe> pf
v.<groupe> := v.<suitemolsym> ;
ar.<suitemolsym> := ar.<groupe> ;

```
<groupe> ::= atome <listesubst>
        stockerval (at.atome, v.<groupe> + v.<listesubst>) ;
        ar.<groupe> := enraciner (at.atome, f.<listesubst>) ;

<listesubst>1 ::= <subst> <M> <listesubst>2
        v.<listesubst>1 := v.<listesubst>2 + (n.<M> * v.<listesubst>2) ;
        f.<listesubst>1 := ajouter (n.<M>, s.<subst>, f.<listesubst>2) ;

<listesubst> ::= liaison <groupe>
        v.<groupe> := rang (l.liaison) ;
        v.<listesubst> := rang (l.liaison) ;
        rangersubst (lier (l.liaison, ar.<groupe>), lastsubst, index) ;
        f.<listesubst> := ajouter (1, index, fvide) ;

<listesubst> ::= Λ
        v.<listesubst> := 0 ;
        f.<listesubst> := fvide ;

<subst> ::= atome
        stockerval (at.atome, 1) ;
        v.<subst> := 1 ;
        rangersubst (lier (simple, enraciner (at.atome), fvide)
                          lastsubst, s.<subst>) ;

<subst> ::= po <L> <groupe> pf
        v.<groupe> := rang (l.<L>) ;
        v.<subst>  := rang (l.<L>) ;
        rangersubst (lier (l.<L>, ar.<groupe>), lastsubst, index) ;

<L> ::= liaison
        l.<L> := l.liaison ;

<L> ::= Λ
        l.<L> := simple ;

<M> ::= entier
        n.<M> := n.entier ;

<M> ::= Λ
        n.<M> := 1 ;
```


3.4. Implantation de la grammaire.

Pour implanter cette grammaire nous utilisons l'analyseur de grammaire LL(1) sur IRIS 80 mis au point par P.Y. CUNIN et M. GRIFFITHS [CUNIN 1978]. Ce produit permet d'insérer des actions sémantiques avec ou sans paramètres. Cependant il ne gère pas la propagation de ces derniers, et il faut l'assurer par ailleurs. La solution adoptée est l'emploi de variables globales à toutes les actions sémantiques.

Pour construire la représentation interne d'une molécule, nous utilisons la technique classique de construction postfixée réalisée à l'aide d'une pile. Il n'existe pas, dans la version PASCAL utilisée, de type correspondant à cette structure de données, et nous l'avons simulée dans un tableau *param* (nous aurions pu également utiliser le type pointeur qui permettait une allocation dynamique des éléments de la pile). Chaque élément de *param* contient une liaison, un atome et une forêt. Le sommet de pile est défini par la variable *sparam*. Nous déclarons donc :

```
const maxparam = 10 ;
var param : tableau [1..maxparam]
              de struct
                   pl : liaison ;
                   pa : atome ;
                   pf : forêt ;
                 fin ;
    sparam : 0..maxparam ;
```

La valeur 0 de *sparam* indique que la pile est vide. L'initialisation et la gestion de *param* sont telles que pour tout i dans [*sparam* + 1, *maxparam*], *param*[i].*pf* = *fvide*. (*param*[i].*pl* et *param*[i].*pa* sont indéterminées).

Une molécule est construite progressivement dans *param*, comme suit :

- *param*[1].*at* reçoit la représentation de l'atome foyer de la molécule ;

- si la molécule est symétrique, *param*[1].*l* reçoit la liaison centre de symétrie, sinon *param*[1].*l* est inutilisée (l'utilisation ou la non utilisation de cette liaison est connue par le contexte syntaxique) ;

- dans *param*[1].*f* sont stockés les indices des substituants de l'atome foyer au fur et à mesure de leur construction.

Les éléments *param[i]*, pour *i* supérieur à 1 sont utilisés pour la construction progressive des substituants, c'est-à-dire que pour tout *i* de 1 à *maxparam*, param[i] est un substituant, en cours de construction, de l'atome *param[i - 1].at*. A la fin de la lecture de ce substituant, il faut le ranger dans le tableau *enssubst*, enlever le sommet de la pile *param* et ajouter son indice à la forêt qui est alors au sommet de pile.

L'indice du nouveau substituant est rangé dans une variable globale *s* déclarée par

var s : 1..maxsubst ;

On utilise également une variable globale *ns* de type *entier* pour stocker le nombre de substituants qui viennent d'être lus ; *ns* est connu par le non-terminal <M>.

Pour un substituant produit par la règle :

<subst> ::= atome

il est inutile d'empiler un élément (*simple, at, fvide*) car on sait que ce substituant est formé d'un atome unique. Nous rangeons simplement le code de l'atome dans une variable globale *as* déclarée par :

var as : 1..maxatom ;

La valence d'un atome *param[i].at* peut être calculée dès que la forêt *param[i].f* est complète car *f* contient alors tous ses substituants et

- si *i* est supérieur à un, *param[i].l* contient la liaison de param[i].at avec l'atome substitué *param[i - 1].at*,

- si *i* est égal à un et si on analyse une molécule non symétrique, les seules liaisons de l'atome sont celles de ses substituants,

- si *i* est égal à un et si on analyse une molécule symétrique, les liaisons de l'atome sont celles de ses substituants et la liaison de symétrie contenue dans *param[1].l*.

La fonction sémantique *stockerval* qui vérifie la valence d'un atome est appelée juste avant la construction et le stockage du substituant ou de la molécule, ce qui évite d'avoir des substituants ou des molécules erronées.

Nous donnons maintenant la liste de toutes les actions sémantiques utilisées et leur rôle.

rsvparam : réservation d'un élément de *param* (*sparam := sparam + 1 ;*).

dépilpar : vidage (*param[sparam].pf := fvide ;*) et enlèvement du sommet de pile (*sparam := sparam + 1 ;*)

svgatome : sauvegarde d'un atome dans *param[sparam].pa*.

svgas : sauvegarde d'un atome dans *as*.

svgentier : sauvegarde d'un entier dans *ns*.

svgun : sauvegarde de la valeur un dans *ns*.

svgliaison : sauvegarde d'une liaison dans *param[sparam].pl*.

svglsimple : sauvegarde d'une liaison simple dans *param[sparam].pl*.

constsubst1 : construction d'un substituant à partir de *param[sparam]* (*sparam > 1*) et stockage dans *enssubst* ; *s* reçoit son indice dans *enssubst*.

constsubst2 : construction d'un substituant à partir de *as* et stockage dans *enssubst* ; *s* reçoit son indice dans *enssubst*.

constmolnonsym : recherche de la représentation canonique d'une molécule écrite sous forme canonique et définie par *param[sparam]* (*sparam = 1* et *param[1].pl* indéterminé) et stockage dans *ensmol*.

constmolsym : construction d'une molécule symétrique à partir de *param[sparam]* (*sparam = 1*) et stockage dans *ensmol*.

ajoutersubst : addition de *ns* substituants identiques, d'indice *s* à la forêt *param[sparam].pf*.

stockerval1 : calcul de la valence de l'atome *param[sparam].pa* à partir de *param[sparam].pl* et *param[sparam].pf*, vérification et stockage dans *valatom*.

stockerval2 : calcul de la valence de l'atome racine d'une molécule non symétrique, vérification et stockage dans *valatom*.

stockerval3 : vérification de la valeur un pour l'atome *as* puis stockage dans *valatom*.

Il reste à insérer les actions sémantiques dans la grammaire.

<réactifs> ::= *rsvparam* <molécule> *dépilpar* <suiteréactifs>

<suiteréactifs> ::= virg <réactifs>
| ptvirg

<molécule> ::= *svgatome* atome <suitemol>
| *svgliaison* liaison <suitemolsym> deux *constmolsym*
| *svglsimple* po <groupe> pf deux *constmolsym*

<suitemol> ::= <listesubst> *stockerval2 constmolnonsym*
| deux *svglsimple stockerval1 constmolsym*

<suitemolsym> ::= *svgatome* atome *stockerval1*
| po <groupe> pf

<groupe> ::= *svgatome* atome <listesubst> *stockerval1*

<listesubst> ::= <subst> <M> *ajoutersubst* <listesubst>
| *rsvparam svgliaison* liaison <groupe>
constsubst1 dépilpar svgun ajoutersubst
| Λ

<subst> ::= *svgas* atome *stockerval3 constsubst2*
| *rsvparam* po <L> <groupe> pf *constsubst1 dépilpar*

<L> ::= *svgliaison* liaison
| *svglsimple*

<M> ::= *svgentier* entier
| *svgun*

Remarque :

L'analyseur lexicographique est en avance d'un mot terminal sur l'analyseur syntaxique. En conséquence nous avons inséré les fonctions sémantiques qui utilisent les informations associées à un mot terminal avant ce mot. C'est le cas des fonctions *svgatome*, *svgas*, *svgentier*, *svgliaison*. La sauvegarde des informations attachées à un mot terminal est faite avant la validation de ce mot, c'est-à-dire avant la reconnaissance syntaxique de ce mot : par exemple on peut sauvegarder dans un élément *param[i].at* l'information attachée à une liaison. Pour éviter de provoquer une erreur du programme par une affectation d'une expression à une variable ayant des types incomparables nous avons d'une part normalisé les informations transmises par l'analyseur lexicographique comme suit :

- un paramètre *code* de type *entier* contient la nature du mot terminal reconnu.
- un paramètre *valeur* de type *entier* contient soit
  - le numéro d'un atome ;
  - le rang d'une liaison ;
  - la valeur d'un entier ;
  - une valeur indéterminée pour les autres terminaux.

D'autre part, nous avons modifié la déclaration de *param* et de *as*

```
var param : tableau [1..maxparam]
                de struct
                    pl,
                    pa : entier ;
                    f : forêt ;
                fin ;
    as : entier ;
```

La vérification syntaxique du code d'un mot terminal est faite après sa sauvegarde. Donc, en cas d'erreur, la pile peut être dans un état incohérent.

Pour être tout à fait rigoureux, il aurait fallu définir une fonction sémantique *svgvaleur* qui sauvegarde le paramètre *valeur* de l'analyseur lexicographique dans une variable auxiliaire *auxi*. La fonction *svgvaleur* est insérée avant le mot terminal et les fonctions *svgatome*, *svgas*, *svgentier*, *svgliaison*, sont insérées après celui-ci. L'information est récupérée après vérification syntaxique à partir de la variable *auxi* et non plus à partir du paramètre *valeur*.

En utilisant l'analyseur de grammaire LL(1), nous avons évité l'écriture fastidieuse des procédures qui reconnaissent les phrases d'une grammaire formelle. Celles-ci sont avantageusement remplacées par une procédure qui interprète la table qu'il produit. Pour une première utilisation, nous n'avons peut-être pas utilisé au mieux toutes ses ressources, notamment pour la spécification des actions sémantiques.

4. STOCKAGE DES PROCESSUS ET CREATION DES PROCESSUS COMPOSES.

Au chapitre II nous avons montré que tous les processus composés de rang n sont obtenus en superposant deux processus simples de rang inférieur ou égal à n dont l'un au moins est de rang n. Dans ce paragraphe nous spécifions la forme du tableau *ensproc* où sont stockés tous les processus élémentaires, et nous donnons les algorithmes pour la construction des processus composés qui utilisent ces relations.

4.1. Structure du tableau *ensproc*.

Tous les processus élémentaires sont stockés dans un tableau *ensproc* dont la taille est donnée par la constante *maxpr*. Les processus élémentaires ont au plus deux réactifs et deux produits. Chaque élément de *ensproc* contient au maximum quatre références à des constituants moléculaires ou radicalaires. D'autre part les processus sont construits mécanisme par mécanisme et modèle par modèle. Pour l'instant, les seules données nécessaires à la classification de ces processus sont les indices des derniers processus de chaque rang et de chaque modèle. Pour définir l'ensemble des processus nous déclarons :

```
type modproc   = (p0, au, ad, de, co, ab, me, di) ;
     molrad    = - maxcnst..maxcnst ;
     processus = tableau [1..4] de molrad ;

var  ensproc   = tableau [1..maxpr] de processus ;
     lastproc  = tableau [modproc, 1..maxméca] de indxpr ;
```

indxpr a déjà été déclaré et représente l'intervalle [*1, maxpr*], et *lastp* contient l'indice du dernier élément de *ensproc*. Pour *i* dans [*1, lastp*] et *j* dans [*1, 4*] nous avons :

- *ensproc*[*i*].[*j*] contient 0 (case vide), l'indice d'un radical ou l'opposé de l'indice d'une molécule.
- *ensproc*[*i*].[*1*] et *ensproc*[*i*].[*2*] contiennent les réactifs et *ensproc*[*i*].[*3*] et *ensproc*[*i*].[*4*] les produits d'un processus élémentaire.

Nous donnons la spécification d'un élément *enspr[i]* pour chaque modèle de processus ; i_c est l'indice dans *ensmol* ou dans *ensrad* du constituant c.

au : r - s ⟶ .r + .s

1	2	3	4
$-i_{r-s}$	0	$i_{.r}$	$i_{.s}$

ad : m + .r ⟶ .m - r

1	2	3	4
$-i_{m}$	$i_{.r}$	$i_{.m-r}$	0

de : .m - r ⟶ m + .r

1	2	3	4
$i_{.m-r}$	0	$-i_{m}$	$i_{.r}$

co : .r + .s ⟶ r - s

1	2	3	4
$i_{.r}$	$i_{.s}$	$-i_{r-s}$	0

ab : m + r - T ⟶ .m - T + .r

1	2	3	4
$-i_{m}$	$-i_{r-T}$	$i_{.m-T}$	$i_{.r}$

me : r - T + .s ⟶ .r + s - T

1	2	3	4
$-i_{r-T}$	$i_{.s}$	$i_{.r}$	$-i_{s-T}$

di : .m - T + .r ⟶ m + r - T

1	2	3	4
$i_{.m-T}$	$i_{.r}$	$-i_{m}$	$-i_{r-T}$

p0 est un modèle de processus fictif.

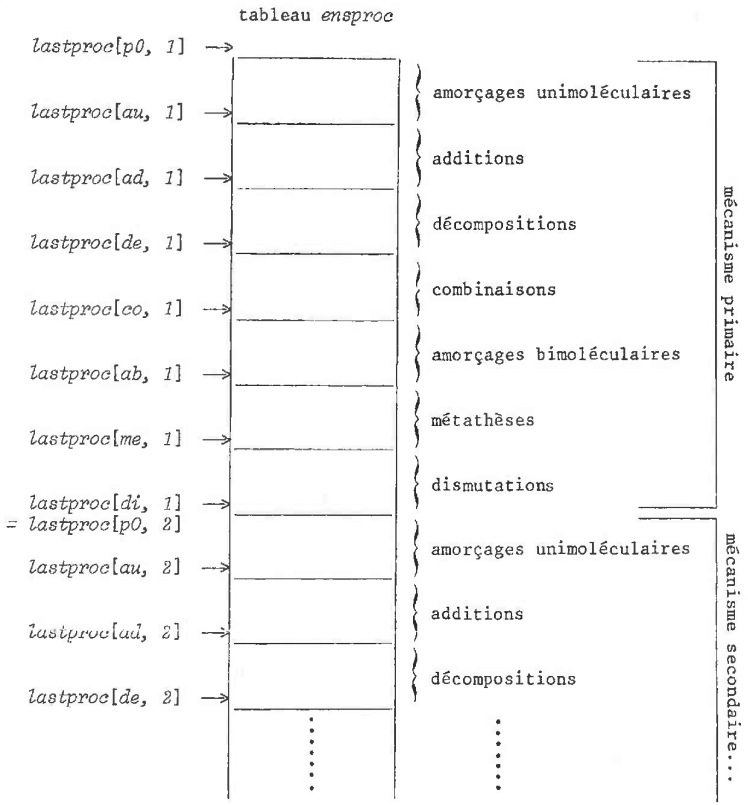
Soient *pr* une variable de type *modproc* et *n* un élément de [*1*, *maxméca*] :

- *lastproc*[*pr*, *n*], *pr* ≠ *p0* est l'indice du dernier processus de rang n et de modèle *pr*.

- *lastproc*[*p0*, *1*] est initialisé à 0.

- *lastproc*[*p0*, *n*], *n* ⩾ *1*, contient la valeur de *lastpr*[*di*, *n* - *1*].

Le "découpage" du tableau *ensproc* par le tableau *lastproc* est représenté ci-dessous :



Les valeurs de *lastproc* sont initialisées dans les procédures *mécaprim* et *mécagen* définies au chapitre II à partir de la variable *lastp*. Il faut y ajouter les instructions nécessaires.

Les amorçages unimoléculaires de rang n sont contenus dans *ensproc*[*i*], pour *i* de *lastproc*[*pred(au) + 1, n*] à lastproc[*au, n*]. Ces éléments sont remplis par la procédure *créerau*. Il en est de même pour les trois autres modèles de processus simples.

Les processus composés sont construits en utilisant uniquement le tableau *ensproc*.

4.2. Création des processus composés.

4.2.1. Principe général.

Nous exposons le principe général de la création des processus composés dans le cas des amorçages bimoléculaires. Puis nous verrons rapidement comment l'adapter dans le cas des métathèses, et des dismutations.

Un amorçage bimoléculaire de forme générale

ab : m + r - T ⟶ .m - T + .r

est la superposition de l'amorçage unimoléculaire

au : r - T ⟶ .r + .T

et de l'addition

ad : m + .T ⟶ .m - T

Soient *ensproc*[*j*] et *ensproc*[*k*] deux éléments de *ensproc* contenant respectivement un amorçage unimoléculaire (au : r - s ⟶ .r + .s) et une addition (ad : m + .u ⟶ .m - u) :

ensproc[*j*]	$-i_{r-s}$	0	$i_{.r}$	$i_{.s}$

ensproc[*k*]	$-i_{m}$	$i_{.u}$	$i_{.m-u}$	0

On peut superposer ces deux processus pour obtenir un amorçage bimoléculaire si

(a) *ensproc*[*j*].[*3*] repère un radical monoatomique
et *ensproc*[*j*].[*3*] = *ensproc*[*k*].[*2*].

ou

(b) *ensproc*[*j*].[*4*] repère un radical monoatomique
et *ensproc*[*j*].[*4*] = *ensproc*[*k*].[*2*].

Dans le premier cas le processus créé est

ab : m + r - s ⟶ .m - r + .s

et l'élément *ensproc*.[*l*] associé à ce processus est obtenu simplement en recopiant successivement *ensproc*[*k*].[*1*], *ensproc*[*j*].[*1*], *ensproc*[*k*].[*3*], *ensproc*[*j*].[*4*]. On obtient de même le processus correspondant au second cas. Il est clair que si les deux radicaux .r et .s sont identiques, les deux superpositions fournissent le même résultat.

Le rang du processus d'amorçage bimoléculaire est donné par la relation :

rang (ab : m + r - T ⟶ .m - T + .r)
= sup {rang (au : r - T ⟶ .r + .T), rang (ad : m + .T ⟶ .m - T)}

Pour chercher les amorçages bimoléculaires de rang 1, il faut superposer les amorçages unimoléculaires de rang 1, contenus dans *ensproc* entre les indices *lastproc*[*pred(au, 1)*] + *1* et *lastproc*[*au, 1*], avec les additions de rang 1 contenues dans *ensproc* entre les indices *lastproc*[*pred(ad, 1)*]+ *1* et *lastproc*[*ad, 1*].

Pour obtenir les amorçages bimoléculaires de rang *n*, supérieur à un, il faut superposer deux processus simples de rang inférieur ou égal à *n*, dont l'un au moins est de rang *n*. Or les amorçages unimoléculaires de rang *k* et de rang *k + 1*, par exemple, ne sont pas contigus dans *ensproc*. La recherche des processus doit être faite en plusieurs étapes :

- pour *k* de *1* à *n - 1*, superposer les amorçages unimoléculaires de rang *k*, contenus dans *ensproc* entre les indices *lastproc*[*pred(au), k*] + *1* et *lastproc*[*au, k*], avec les additions de rang *n* contenues dans *ensproc* entre les indices *lastproc*[*pred(ad), n*] + *1* et *lastproc*[*ad, n*]

- pour *k* de *1* à *n*, superposer les amorçages unimoléculaires de rang *n*, contenus dans *ensproc* entre les indices *lastproc*[*pred(au), n*] + *1* et

lastproc[au, n], avec les additions de rang *k*, contenues dans *ensproc* entre les indices *lastproc[pred(ad), k] + 1* et *lastproc[ad, k]*.

Les métathèses (me : r - T + .s ⟶ .r + s - T) sont obtenues de la même façon en superposant un amorçage unimoléculaire (au : r - T ⟶ .r + .T) et une combinaison (co : .s + .T ⟶ s - T), et les dismutations (di : .m - T + .r ⟶ m + r - T) en superposant une décomposition (de : .m - T ⟶ m + T) et une combinaison (co : .r + .T ⟶ r - T).

Nous utilisons une même procédure *abmediens* pour rechercher les processus composés des trois modèles. Son interface est la suivante :

procédure abmediens (in p1, p2 : modproc ;
in k1, k2 : indxméca ;
inout lastp : indxpr)

Cette procédure cherche toutes les superpositions entre les processus de modèle *p1* et de rang *k1*, et ceux de modèle *p2* et de rang *k2*. Le tableau suivant indique la valeur des paramètres d'entrée en fonction de la nature des processus cherchés.

processus cherchés		*p1*	*p2*	*k1*	*k2*
modèle	rang				
ab	*1*	*au*	*ad*	*1*	*1*
	n > 1			*1 à n-1*	*n*
				n	*1 à n*
me	*1*	*au*	*co*	*1*	*1*
	n > 1			*1 à n-1*	*n*
				n	*1 à n*
di	*1*	*de*	*co*	*1*	*1*
	n > 1			*1 à n-1*	*n*
				n	*1 à n*

lastp est l'indice du dernier élément de *ensproc*

Les instructions d'appel à la procédure *abmediens* doivent être insérées dans les procédures *mécaprim* et *mécagen*. Avant de donner son algorithme, il faut caractériser les radicaux monoatomiques, et chercher les processus simples qu'il faut superposer.

4.2.2. Caractérisation des radicaux monoatomiques.

Dans le tableau *ensproc* les constituants sont repérés par leurs indices. Ceux-ci sont affectés au fur et à mesure de leur production. En particulier, les radicaux monoatomiques sont engendrés par les amorçages unimoléculaires du mécanisme primaire, et leurs indices dans *ensrad* sont mêlés à ceux des autres radicaux libres. A priori, pour savoir si un radical d'indice *i* est monoatomique, il faut accéder à l'élément *ensrad[i]* et tester si sa composante *f* est vide, ce qui est très lourd.

Cependant les radicaux monoatomiques présents dans un mécanisme réactionnel sont tous des radicaux primaires de la forme .T où T est un atome monovalent présent dans les réactifs. Après la lecture de ces derniers, on connaît, grâce au tableau *valatom*, l'ensemble de tous les atomes et leur valence, ce qui permet d'initialiser *ensrad* avec les radicaux monoatomiques. Ainsi si *nbatvl* est le nombre des atomes monovalents, les radicaux monoatomiques ont des indices compris entre *1* et *nbatvl*. Donc un élément *ensproc[i].[j]* du tableau *ensproc* repère un radical monoatomique, si et seulement si $1 \leqslant ensproc[i].[j] \leqslant nbatvl$.

Ce test est plus rapide que le premier test annoncé. En contrepartie, sa mise en oeuvre ne demande qu'une lecture du tableau *valatom* et l'initialisation de *ensrad*. Nous aurions pu éviter la lecture de *valatom* en construisant les radicaux monoatomiques au fur et à mesure de la lecture d'un nouvel atome monovalent en utilisant la procédure *stockerval*, mais cela aurait nui à la structure du programme et à sa lisibilité.

4.2.3. Recherche des processus simples à superposer.

Dans ce paragraphe nous notons

infi = lastproc[pred(p1), k1] + 1
supi = lastproc [p2, k2]
i = 1, 2

La recherche des processus simples à superposer s'effectue en deux étapes :

- on cherche dans *ensproc*, entre les indices *inf1* et *sup1*, tous les processus *ensproc*[*j*] qui produisent un radical monoatomique que nous appelons *lien* ;

- pour chaque processus *ensproc*[*j*] trouvé, on cherche entre les indices *inf2* et *sup2*, un processus *ensproc*[*k*] dont *lien* est un réactif.

Nous traitons chaque modèle de processus simple séparément.

4.2.3.1. Les amorçages unimoléculaires.

Les amorçages unimoléculaires de rang *k* qui produisent des radicaux monoatomiques sont répartis dans toute la tranche du tableau comprise entre les indices *inf1* et *sup1*. Il faut donc les examiner tous. D'après l'algorithme utilisé pour la procédure *auelt* qui crée ces processus (cf. § 2.3. du chapitre IV), si un radical monoatomique est produit par le processus *ensproc*[*i*], il est rangé dans *ensproc*[*i*].[*3*]. Il faut envisager le cas d'un amorçage unimoléculaire appliqué à une molécule diatomique qui produit deux radicaux monoatomiques distincts rangés respectivement dans *ensproc*[*i*].[*3*] et *ensproc*[*i*].[*4*].

4.2.3.2. Les décompositions.

De même que pour les amorçages unimoléculaires, il faut examiner toutes les décompositions de *ensproc* entre *inf1* et *sup1*. Le radical produit par une décomposition *ensproc*[*i*] est rangé dans *ensproc*[*i*].[*4*].

4.2.3.3. Les additions.

Il faut chercher les additions contenues dans *ensproc* entre les indices *inf2* et *sup2*, dont le radical référencé par *lien* est réactif. Le réactif radicalaire d'une addition *ensproc*[*i*] est rangé dans *ensproc*[*i*].[*2*]. L'algorithme choisi pour la procédure *adens* (§ 3.4.2. chapitre II) entraîne que l'addition des radicaux r_1, r_2, r_3, ... sur les molécules m_1, m_2, ..., sont traitées dans l'ordre suivant :

$m_1 + r_1 \longrightarrow .m_1 - r_1$

$m_1 + r_2 \longrightarrow .m_1 - r_2$

$m_1 + r_3 \longrightarrow .m_1 - r_3$

.....

$m_2 + r_1 \longrightarrow .m_2 - r_1$

$m_2 + r_2 \longrightarrow .m_2 - r_2$

$m_2 + r_3 \longrightarrow .m_2 - r_3$

.....

Dans la procédure *adens*, si l'on remplace les instructions :

```
pour i := mininf à minsup faire
  pour j := rininf à rinsup faire
    ......
  fpour ;
fpour ;
```

par les instructions

```
pour j := rininf à rinsup faire
  pour i := mininf à minsup faire
    ......
  fpour ;
fpour ;
```

les additions sont traitées dans l'ordre suivant :

$m_1 + r_1 \longrightarrow .m_1 - r_1$

$m_2 + r_1 \longrightarrow .m_2 - r_1$

.....

$m_1 + r_2 \longrightarrow .m_1 - r_1$

$m_2 + r_2 \longrightarrow .m_2 - r_2$

.....

$m_1 + r_3 \longrightarrow .m_1 - r_3$

$m_2 + r_3 \longrightarrow .m_2 - r_3$

.....

D'autre part l'initialisation du tableau *ensrad* avec les radicaux monoatomiques entraîne que ceux-ci ont des indices compris entre *1* et *nbatvl*, et la recherche des additions de rang *n* commence toujours par l'addition des radicaux de rang un avec les molécules de rang *n*. Donc en faisant la transformation donnée dans

la procédure *adens*, nous trouverons dans *ensproc*, à partir de l'indice *sup2*, les additions du radical *ensrad[1]*, puis celles du radical *ensrad[2]*, etc... La recherche des processus d'addition est donc localisée aux premiers processus rencontrés.

4.2.3.4. Les combinaisons.

La combinaison de deux radicaux libres existe toujours et elle est unique. En conséquence on peut calculer à l'aide des tableaux *lastrad* et *lastproc* les indices des processus de combinaisons faisant réagir un radical monoatomique d'indice *lien*. Nous traitons d'abord le cas des combinaisons primaires, ensuite le cas des combinaisons de rang supérieur à un.

a) Cas des combinaisons primaires.

Notons $f(k, m)$ la somme des $k + 1$ termes

$m, m-1, m-2, \ldots, m-k$

définie pour $m \geqslant 0$ et $0 \leqslant k \leqslant m$.

$$f(k, m) = km - \frac{k(k-1)}{2}$$

Si $m = lastrad[1]$, numéro du dernier radical primaire, et si $l = lastproc[pred(co), 1]$, on peut vérifier que $l + f(k, m)$ est l'indice dans *ensproc* du dernier processus de combinaison qui fait réagir le radical d'indice k, $k \geqslant 1$.

Exemple :

Considérons $m = 4$ radicaux primaires, $.r_1$, $.r_2$, $.r_3$ et $.r_4$ et les combinaisons de ces radicaux rangées dans *ensproc* :

	1	2	3	4	
$l+1$	1	1	$-i_{r_1 - r_1}$	0	
$l+2$	1	2	$-i_{r_1 - r_2}$	0	
$l+3$	1	3	$-i_{r_1 - r_2}$	0	
$l+4$	1	4	$-i_{r_1 - r_4}$	0	$l + f(1, 4)$
$l+5$	2	2	$-i_{r_2 - r_2}$	0	
$l+6$	2	3	$-i_{r_2 - r_3}$	0	
$l+7$	2	4	$-i_{r_2 - r_4}$	0	$l + f(2, 4)$
$l+8$	3	3	$-i_{r_3 - r_4}$	0	
$l+9$	3	4	$-i_{r_3 - r_4}$	0	$l + f(3, 4)$
$l+10$	4	4	$-i_{r_4 - r_4}$	0	$l + f(4, 4)$

On peut montrer que les processus d'indices $j = l + f(i, m) + lien - i$, i de 0 à $lien - 2$, sont tels que $ensproc[j].[2] = lien$, et que les processus d'indices j, j dans $[l + f(lien - 1, m) + 1, l + f(lien, m)]$ sont tels que $ensproc[j].[1] = lien$.

Ces relations donnent directement les indices des combinaisons primaires faisant réagir un radical monoatomique donné.

b) Cas des combinaisons de rang supérieur à un.

Considérons deux radicaux monoatomiques, .X, .Y d'indices 1 et 2, et trois radicaux libres t_1, t_2, t_3, de rang n supérieur à 1 d'indices a, a + 1 et a + 2.

Nous avons représenté ci-dessous une partie du tableau *ensproc* à partir de l'indice $l + 1 = lastproc[pred(co), n] + 1$

	1	2	3	4
$l + 1$	1	a	$-i_{t_1} - X$	0
$l + 2$	1	a + 1	$-i_{t_2} - X$	0
$l + 3$	1	a + 2	$-i_{t_3} - X$	0
$l + 4$	2	a	$-i_{t_1} - Y$	0
$l + 5$	2	a + 1	$-i_{t_2} - Y$	0
$l + 8$	2	a + 2	$-i_{t_3} - Y$	0

Soit $m = lastrad[n] - lastrad[n-1]$ le nombre de radicaux libres de rang n. On établit aisément que les processus d'indices j compris entre $l + (lien - 1) * m + 1$ et $l + lien * m$ sont tels que $ensproc[j].[1] = lien$.

Nous pouvons donc calculer directement l'indice des processus de combinaisons faisant réagir un radical monoatomique d'indice *lien*.

4.2.4. Algorithme.

La procédure *abmediens*, lorsqu'elle trouve un processus $ensproc[j]$ de modèle *p1* (*p1* = *au* ou *de*) appelle la procédure *abmedielt* qui cherche les processus de modèle *p2* (*p2* = *ad* ou *co*) qui peuvent s'associer à $ensproc[j]$ pour créer un processus composé. L'interface de *abmedielt* est :

```
procédure abmedielt (in p2 : modproc ; in k2 : indxméca ;
                     in lien, réactif, produit : molrad ;
                     inout lastp : indxpr) ;
```


k2 est le rang des processus de modèle *p2* qu'il faut associer au processus :

produit ⟶ *réactif* + *lien*

lien désigne le radical monoatomique ;

lastp est l'indice du dernier processus créé.

La création d'un processus composé est faite par la

```
procédure créerabmedi (in ir1, ir2, ip1, ip2 : molrad ;
                       inout lastp : indxpr) ;
```

ir1 et *ir2* sont les indices des réactifs, *ip1* et *ip2* sont ceux des produits.

```
procédure abmediens (in p1, p2 : modproc ;
                     in k1, k2 : indxméca ;
                     inout lastp : indxpr) ;

var lien, produit, réactif : molrad ;
  i : indxpr ;
  proci : processus ;

début
  cas p1 de
    au : début
           pour i := lastproc[pred(p1), k1] + 1
               à lastproc[p1, k1] faire
             proci := ensproc[i] ;
             si proci[3] ≤ nbatv1
               alors % c'est un lien %
                 abmedielt (p2, k2, proci[3], proci[1], proci[4],
                            lastp) ;
             fsi ;
             si proci[4] ≤ nbatv1 et proci[3] ≠ proci[4]
               alors % c'est un lien %
                 abmedielt (p2, k2, proci[4], proci[1], proci[3],
                            lastp) ;
             fsi ;
           fpour ;
         fin ;
```

```
    de : début
          pour i := lastproc[pred(p1), k1] + 1
               à lastproc[p1, k1] faire
            proci := ensproc[i] ;
            si proci[4] ≤ nbatv1
              alors % c'est un lien %
                abmedielt (p2, k2, proci[4], proci[1], proci[3],
                lastp) ;
            fsi ;
          fpour ;
        fin ;
  fcas ;
fin ; % abmediens %


procédure abmedielt (in p2 : modproc ; in k2 : indxméca ;
                     in lien, produit, réactif : molrad ;
                     inout lastp : indxpr) ;

var i, l, j, dernier : indxpr ;
  m : indxcnst ;
  proci : processus ;

début
  cas p2 de
    ad : début
                % recherche de la première addition avec lien %
         i := lastproc[pred(p2), k2] + 1 ;
         dernier := lastproc[p2, k2] ;
         tantque ensproc[i].[2] ≠ lien et i ≤ dernier faire
           i := i + 1 ;
         ftantque ;
           % recherche de la dernière addition avec lien et
                                              superposition  %
         proci := ensproc[i];
         tantque proci[2] = lien et i ≤ dernier faire
           créerabmedi (proci[1], produit, réactif, proci[3],
                        lastp) ;
           i := i + 1 ;
           proci := ensproc[i] ;
         ftantque ;
```



```
    co : début
          l : lastproc[pred(p2), k2] ;
          si k2 = 1
            alors   % on cherche les combinaisons primaires %
              m := lastrad[1] ;
              pour i := 0 à lien - 2 faire
                proci := ensproc[l + (i * m) - (i * (i - 1) div 2) + lien - i]
                % proci[2] = lien %
                créerabmedi (produit, proci[1], réactif, proci[3],
                             lastp) ;
              fpour ;
              pour  i := l + (lien - 1) * m - ((lien - 1) * (lien - 2 div 2) + 1
                à  l + (lien * m) - (lien * (lien - 1) div 2) faire
                  proci := ensproc[i] ;  % proci[1] = lien %
                  créerabmedi (produit, proci[2], réactif, proci[3],
                               lastp) ;
              fpour ;
            sinon  % on cherche les combinaisons de rang n > 1 %
              m := lastrad[k2] - lastrad[k2 - 1] ;
              pour j := l + (lien - 1) * m + 1 à l + lien * m faire
                % ensproc[j].[1] = lien %
                créerabmedi (produit, ensproc[j],[2], réactif,
                             ensproc[j].[3], lastp) ;
              fpour ;
          fsi ;
        fin ;
  fcas ;
fin ; % abmediens %
```

Avec ces procédures nous avons terminé la description du programme *mécanisme*. Les processus composés sont tous des processus ayant deux réactifs, et leur nombre est très important. Bien qu'ils soient construits à partir des processus simples, leur présence est nécessaire dans un mécanisme, car sans eux on ne peut poursuivre la modélisation d'une réaction.

Pour l'instant nous avons réalisé deux versions du programme, pour tester les algorithmes. L'une recherche le mécanisme de rang n, où seuls sont créés les processus simples. L'autre crée tous les processus primaires, y compris les processus composés.

C O N C L U S I O N

Dans ce projet, nous avons voulu répondre aux besoins des chimistes, en essayant de concevoir avec eux des outils informatiques utiles pour leurs travaux. Comme dans toute collaboration, il y a un problème de dialogue et de compréhension entre des personnes travaillant dans des spécialités différentes. La première étape du travail a consisté en de nombreuses réunions au cours desquelles nous avons amené les chimistes à formuler leurs problèmes de manière aussi précise que possible.

A ce stade le problème a été défini par un ensemble de sept modèles de processus élémentaires, qu'il faut appliquer à une liste de molécules données pour construire un mécanisme réactionnel. Ces modèles permettent de modéliser une partie des réactions radicalaires en phase gazeuse, domaine principal de recherche du laboratoire de Cinétique Chimique. Les constituants qui entrent en jeu dans ces réactions sont les molécules et les radicaux libres. L'application des modèles de processus est mal définie pour les composés cycliques et pour les radicaux ayant plus d'un électron célibataire. En outre, elle ne modifie pas la valence d'un atome dans un constituant. Nous avons donc restreint le domaine d'application aux constituants non cycliques, ayant au plus un électron célibataire et dans lesquels un atome donné a une valence fixe. Ceci revient à éliminer les réactions des cycles, les mécanismes faisant intervenir les biradicaux, ainsi que les composés comportant des éléments à valences variables, tels que l'oxygène. Pour ces constituants, nous avons défini une notation linéaire, proche de celles habituellement utilisées par les chimistes. La grammaire formelle de cette notation est relativement simple. La description d'un constituant ne reflète pas tous ses aspects stéréochimiques, mais elle est suffisante pour notre problème. La similitude entre la notation externe et la représentation interne limite l'interface nécessaire pour passer de l'une à l'autre. Le développement théorique important que nous avons présenté pour la recherche d'une notation canonique a, d'une part,

fourni un algorithme non itératif qui transforme toute notation proposée par un utilisateur en notation canonique, et d'autre part, facilite la création des processus. Cet algorithme utilisant principalement la théorie des graphes peut avoir d'autres applications que la représentation des molécules.

Nous avons dû apporter une limitation pour créer un nombre fini de processus élémentaires. Pour une première version, nous avons choisi la solution la plus simple à mettre en oeuvre qui consiste à majorer la taille des radicaux produits par les processus d'addition. Ceci élimine la possibilité de modéliser des réactions de polymérisation.

Moyennant ces quelques restrictions, nous avons réalisé une première version qui comme tout prototype présente des imperfections. Nous avons principalement résolu les problèmes théoriques qui se posaient et cette première version doit être considérée comme un premier résultat concret à partir duquel peut se poursuivre la collaboration.

La principale critique qu'on peut faire à ce prototype est de produire beaucoup trop de processus. Mais un chimiste peut éliminer d'emblée une grande partie des processus qu'il juge négligeables. Une solution rapide à mettre en oeuvre consiste en un petit programme de dialogue qui permette de faire aisément ces premières simplifications. Ce dialogue peut également servir à compléter le mécanisme par les constantes cinétiques qui peuvent dans un premier temps être entrées "à la main". Une commande spécifique, à utiliser lorsque le mécanisme est jugé satisfaisant, doit être prévue pour stocker sur fichier toutes les informations nécessaires à la simulation du mécanisme, et notamment la matrice des coefficients stoechiométriques qui est à la base du traitement numérique.

Ce dialogue réalisé, les chimistes pourront évaluer les performances et l'apport d'un tel système. Ce n'est qu'après un nombre important d'utilisations qu'on sera à même de juger si parmi les restrictions imposées aux constituants, certaines sont justifiées ou trop contraignantes, et si leur formulation est adaptée pour que les utilisateurs prennent conscience des limites du système.

Comme nous l'avons déjà signalé, le prototype tel qu'il est réalisé actuellement présente une grosse lacune quant à l'aide apportée aux chimistes

pour réduire le nombre de processus élémentaires créés. Les constantes cinétiques sont, en quelque sorte, une mesure statistique de la facilité d'un processus donné. Elles permettent donc d'établir quels sont les processus déterminants et quels sont les processus négligeables. Mais leur détermination, faite expérimentalement, fait encore l'objet d'importants travaux de recherche en cinétique chimique. Cependant les travaux de BENSON sur l'évaluation des constantes cinétiques à partir de la structure des molécules semblent apporter une solution intéressante au problème, et des recherches dans ce sens sont engagées dans le cadre de la collaboration entre le D.C.P.R. et le C.R.I.N. .

Nous avons vu que l'ensemble des modèles de processus fixé au départ limitait le champ d'application du système à un certain type de réactions. Une extension peut donc être obtenue en introduisant de nouveaux modèles. Il serait certes souhaitable que le chimiste puisse en sélectionner une partie en fonction du type de réaction à traiter.

Une étude intéressante serait de voir quelles sont les analogies existant entre notre approche et celle de la synthèse organique assistée par ordinateur.

B I B L I O G R A P H I E

[ALRAN 1979]

D. ALRAN, G.M. CÔME, P.Y. CUNIN et M. GRIFFITHS
Mechanistic modelling of homogeneous reactors : a chemical compiler.
Comp. & Chem. Eng., 3, p. 87-89, 1979.

[AZAY 1981]

P. AZAY
Modélisation et simulation mécanistiques de réactions radicalaires complexes. Exemple de la pyrolyse du néopentane vers 700°C.
Thèse d'Etat, I.N.P.L.-E.N.S.I.C., Nancy, juillet 1981.

[BARONNET 1971, 1974]

F. BARONNET, M. DZIERZYNSKI, G.M. CÔME, R. MARTIN et M. NICLAUSE
The pyrolysis of neopentane at small extents of reaction.
Intern. J. Chem. Kin., 3, p. 197-213, 1971.

F. BARONNET, G.M. CÔME et M. NICLAUSE
Inhibition et auto-inhibition de la pyrolyse du néopentane par l'isobutène.
J. Chim. Phys., 71, p. 1214-1222, 1974.

[BENSON 1976]

S.W. BENSON
Thermochemical kinetics. Methods for the estimation of thermodynamical data and rate parameters. Second edition.
Wiley Interscience, 1976.

[BERGE 1971]

C. BERGE
Graphes et hypergraphes.
Dunod, 1971.

[BONNET 1979]
P. BONNET, J.C. DERNIAME, M.H. ZIMMER, F. CHOPLIN et G. KAUFMANN
Stratégie en synthèse chimique assistée par ordinateur.
Comp. Sc., 13, p. 287-308, 1979.

[BURNETT 1954]
G.M. BURNETT
Mechanism of polymer reactions.
Interscience, New York and London, 1954.

[CÔME 1980, 1981]
Dans Techniques de l'Ingénieur :

G.M. CÔME et M. NICLAUSE
Cinétique chimique - Introduction.
Fascicule J 1100, 1980.

G.M. CÔME et M. NICLAUSE
Cinétique chimique - Généralités.
FasciculesJ 1110 et J 1111, 1980.

G.M. CÔME, R. MARTIN, M. NICLAUSE et F. BARONNET
Cinétique en phase gazeuse - II. Réactions complexes.
Fascicule J 1120, 1981.

G.M. CÔME
Cinétique chimique appliquée - Expérimentation.
Fascicule J 1130, 1981.

G.M. CÔME
Cinétique chimique appliquée - Analyse stoechiométrique et cinétique.
Fascicule J 1132, 1981.

G.M. CÔME
Cinétique chimique appliquée - Modélisation.
Fascicule J 1134, 1981.

[COREY 1972]
E.J. COREY, W.T. WIPKE, R.D. CRAMER III et W.J. HOWE
Computer-assisted synthetic analysis. Facile man-machine communication of chemical structure by interactive computer graphics.
J. Am. Chem. Soc., 94, p. 421-430, 1972.

[CUNIN 1978]
P.Y. CUNIN et M. GRIFFITHS
Générateurs d'analyseurs LL(1) sur IRIS 80.
C.R.I.N., 78-R-008, février 1978.

[EDELSON 1976]
D. EDELSON
A simulation language and compiler to aid computer solution of chemical kinetics problems.
Comp. & Chem., 1, p. 29-33, 1976.

[FOSDICK 1975]
L.D. FOSDICK, L.J. OSTERWEIL
Validation and global optimization of programs.
4th Texas Conference on Computing Systems (U.S.A.), novembre 1975.

[FOSTER 1968]
J.M. FOSTER
A syntax improving device.
Computer Journal, mai 1968.

[GOLDFINGER 1948]
P. GOLDFINGER, M. LETORT et M. NICLAUSE
Relations entre le mécanisme et l'ordre d'une réaction ; règles gouvernant l'initiation et la rupture des chaînes dans la pyrolyse homogène des vapeurs organiques.
Volume commémoratif Victor Henri : "Contribution à l'étude de la structure moléculaire", p. 283-296, Desoer, Liège, 1948.

[GRIFFITHS 1974]
M. GRIFFITHS
LL(1) Grammars and analysers.
Compiler construction - An advanced course.
Lecture Notes in Computer Science n° 21, p. 57-84.
Springer Verlag, 1974.

[GROSS 1967]
M. GROSS et A. LENTIN
Notions sur les grammaires formelles.
Gauthier Villard, Paris, 1967.

[INGOLD 1953]
C.K. INGOLD
Structure and mechanism in organic chemistry.
Ithaca, N.Y., Cornell University Press, 1953.

[KNUTH 1971a]
D.E. KNUTH
Examples of formal semantics.
Symposium on semantics of algorithmic languages.
Lecture Notes in Mathematics n° 188, p. 212-235.
Springer Verlag, 1971.

[KNUTH 1971b]
D.E. KNUTH
Top down analysis.
Acta Informatica, 1971.

[LAFONT 1977]
S. LAFONT
Compilateur de langage chimique pour le traitement de modèles en cinétique chimique.
Rapport de stage, E.N.S.I.M.A., Grenoble, 1977.

[LORHO 1978]
B. LORHO
Semantics attributes processing in the system DELTA.
State of the art and future trends in compilation, Ecole CREST, Montpellier, édité par l'I.R.I.A., 1978.

[LOZAC'H 1968]
N. LOZAC'H
Règles de nomenclature pour la chimie organique (Règles 1965) élaborées par la Commission de Nomenclature en Chimie Organique de l'I.U.P.A.C.
Soc. Chim. Fr., 1968.

[MARCHAND 1979]
P. MARCHAND
Théorie des graphes.
Cours du CI d'informatique, Nancy I, partie algèbre, chap. 3, C.R.I.N., 79-E-020, 1979.

[MARQUAIRE 1978]
P.M. MARQUAIRE et G.M. CÔME
Non quasi-stationary state pyrolysis : kinetic parameters of neopentane pyrolysis mechanism.
Reaction Kinetics and Catalysis Letters, 9, p. 171-175, 1978.

[MAURICE 1978]
P. MAURICE
PASCAL 80 : Manuel d'utilisation.
Service de synthèse et d'orientation de la recherche informatique, 1978.

[METZGER 1973]
J. METZGER
Mécanismes réactionnels, 13, p. 1006-1010, 1973.

[NICLAUSE 1966]
M. NICLAUSE, R. MARTIN, F. BARONNET et G. SCACCHI
Etude théorique d'un mécanisme d'accélération ou d'inhibition de réactions en chaînes de décomposition.
Revue de l'Institut Français du Pétrole, 21, p. 1724-1760, 1966.

[RONDEAU 1976]
J.A. RONDEAU, G.M. CÔME et J.F. LARGE
Pyrolysis of 2,2 dimethylpropane in a continuous flow stirred tank reactor.
Industrial and Laboratory Pyrolyses (L.F. ALBRIGHT et B.L. CRYNES, ed.) A.C.S. Symp. Ser., 32, p. 131-151, 1976.

[SMITH 1975]
E.G. SMITH et P.A. BAKER
The Wiswesser line-formula chemical notation (WLN).
Third edition.
CIMI, 1975.

[VILLERMAUX 1980]
J. VILLERMAUX
Réacteurs chimiques.
Techniques de l'Ingénieur, fascicule J 1040, 1980.

[WIPKE 1974]
W.T. WIPKE, S.R. HELLER, R.J. FELDMANN et E. HYDE
Computer representation and manipulation of chemical information.
Wiley Interscience, 1974.

[WIRTH 1971]
N. WIRTH
The programming language PASCAL.
Acta Informatica, p. 35-63, 1971.

institut national polytechnique de lorraine

Le Président,

N/Réf. : Scol.

AUTORISATION DE SOUTENANCE DE THESE DE DOCTORAT-D'INGENIEUR

-:-:-:-:-:-:-:-:-:-:-:-:-:-

VU LE RAPPORT ETABLI PAR :

Monsieur le Professeur COME G.M.

le Président de l'Institut National Polytechnique de Lorraine autorise :

Madame VOGIN Laurence née HAUX

à soutenir, devant l'I.N.P.L., une thèse intitulée :

"MISE EN OEUVRE INFORMATIQUE DE LA MODELISATION DE REACTIONS CHIMIQUES"

en vue de l'obtention du titre de DOCTEUR-INGENIEUR

Spécialité *"INFORMATIQUE"*

Fait à NANCY, le *13 Septembre 1982*

Le Président de l'I.N.P.L.
p/o le Vice-Président de l'I.N.P.L.

M. LUCIUS
K. HOUPERT