UNIVERSITÉ DE NEUCHATEL - FACULTÉ DES SCIENCES

BARBARA

ANALYSE DE DONNÉES INFORMELLES A L'AIDE DE RÉSEAUX SYSTÉMIQUES

Thèse
prèsentée à la Faculté des Sciences
par

FRANÇOIS GRIZE

licenciè ès sciences
pour l'obtenlion du grade de
docteur ès sciences

- 1981 -

IMPRIMATUR POUR LA THÈSE

BARBARA: Analyse de données informelles à l'aide de réseaux systémiques

de Monsieur François Grize

UNIVERSITÉ DE NEUCHATEL

FACULTÉ DES SCIENCES

La Faculté des sciences de l'Université de Neuchâtel, sur le rapport des membres du jury,

Messieurs P. Banderet, A. Strohmeier, J. Ogborn (Londres) et J.-C. Derniame (Nancy)

autorise l'impression de la présente thèse.

Neuchâtel, le 25 février 1981

Le doyen:

K. Bernauer

TABLE DES MATIERES

CHAPITRE CINQUIEME: COMMENT UTILISER BARBARA

RESUME

Le premier chapitre de ce travail est consacré à la linguistique systémique. Après avoir souligné les principaux liens qui peuvent ou ont pu exister entre la linguistique et l'informatique, il est mis en évidence la manière dont la linguistique systémique étudie le langage et les langues naturelles. En particulier, on verra qu'elle considère le langage comme un ensemble de "systèmes". Ces systèmes peuvent se combiner entre eux pour former ce que l'on appelle un réseau systémique. Le formalisme utilisé pour représenter un réseau systémique est à la base de la méthode présentée ici, méthode qui permet l'analyse de données informelles.

Le deuxième chapitre décrit de manière "algébrique" ce qu'est un réseau systémique. Le formalisme utilisé pour cette description est emprunté à la théorie des grammaires formelles. En particulier, on montrera que tout langage indépendant de contexte peut s'exprimer à l'aide d'un réseau systémique.

Dans le troisième chapitre, on examine comment, à partir de réseaux systémiques, il est possible d'analyser des données informelles, comme par exemple: des interviews, des résultats d'enquête, des questions d'examen, etc... En fait, on verra que les réseaux systémiques permettent de décrire des catégories de complexité arbitraire. La méthode consiste donc à produire un "schéma" général (réseau systémique) de données et à "coder" ces dernières à l'aide des constituants de ce schéma. Un exemple complet d'utilisation de la méthode est traité en détail.

Les deux derniers chapitres, qui constituent la partie essentielle de ce travail, sont consacrés à un programme appelé BARBARA qui rend la méthode utilisable sur ordinateur. Ce programme est capable de lire n'importe quel réseau systémique; il en vérifie la cohérence et en garde une représentation en mémoire. Dans une deuxième phase, il accepte des "phrases" correspondant à la codification des données informelles. Il vérifie que chaque phrase est un chemin possible dans le réseau et mémorise chaque chemin emprunté. Enfin, il réalise des fonctions d'analyse et de comptage, fonctions qui s'expriment à l'aide d'expressions de nature ensembliste.

Le quatrième chapitre décrit le programme en détail, tant du point de vue de l'algorithmique que du point de vue de la structure et de l'organisation des données.

Enfin, le dernier chapitre contient un mode d'emploi de BARBARA, illustré par des exemples commentés.

Les nombres entre parenthèses qui se trouvent dans le texte renvoient à des références bibliographiques situées à la fin de chaque chapitre.

CHAPITRE PREMIER

LINGUISTIQUE SYSTEMIQUE

1.1 Linguistique et informatique

On s'accorde généralement à considérer que la linguistique a acquis le statut d'étude scientifique du langage avec F. DE SAUSSURE (1916) (1). Il convient par ailleurs de noter que la linguistique est l'un des domaines scientifiques qui a le plus évolué à cause de l'apparition des ordinateurs.
L'ordinateur étant une machine qui n'accepte pas les définitions imprécises, il est compréhensible que cette évolution se soit faite tout naturellement dans le sens d'une formalisation plus rigoureuse.
Le travail présenté ici s'appuie sur un formalisme (les réseaux systémiques) utilisé par certains linguistes pour étudier et expliquer le langage et les langues naturelles.
Les liens qui ont pu ou peuvent exister entre la linguistique et l'informatique sont nombreux et divers. Dans ce paragraphe, nous allons tenter d'en souligner les principaux aspects.

Linguistique et langages de programmation.

Dans les années cinquante, les "informaticiens" étaient confrontés à un problème fondamental qui consistait à définir et à implanter des langages permettant la programmation des ordinateurs. A cette même époque, avec N. CHOMSKY (2), paraissaient les premiers travaux des linguistes sur la formalisation de la syntaxe.

Les langages de programmation d'alors (essentiellement FORTRAN), quand bien même artificiels - par opposition aux langues naturelles - n'étaient définis que de manière informelle, ce qui conduisait à certaines difficultés d'interprétation. De plus, il n'y avait aucune théorie traitant de leur compilation, c'est-à-dire permettant la réalisation d'automates capables de dire si une chaîne donnée appartient ou non au langage concerné.

ALGOL 60 est le premier langage de programmation qui va bénéficier directement des travaux de N. CHOMSKY. En effet, le

rapport, paru en 1960 (3), qui définit ce langage, en précise de manière formelle tout l'aspect syntaxique.

Parallèlement, à la même époque, s'élabore une théorie des automates, c'est-à-dire de modèles mathématiques, correspondant aux différents types des langages de N. CHOMSKY. Toute la théorie de la compilation va se fonder sur ces modèles, qui sont relativement faciles à implanter sur ordinateurs.

La traduction automatique.

Pour comprendre l'effort important qui a été fait dans le domaine de la traduction automatique, il faut se rapporter au contexte historique des années cinquante. Cette époque se caractérise par ce que l'on a coutume d'appeler la "guerre froide". L'un des problèmes du gouvernement américain consistait à accumuler le plus d'informations possibles sur l'antagoniste, c'est-à-dire l'U.R.S.S. Pour ce faire, il convenait de traduire l'ensemble des journaux parus en russe, ce qui dépassait de beaucoup les possibilités des rares citoyens américains parlant cette langue. Par ailleurs, les ordinateurs donnaient la preuve qu'ils étaient capables de manipuler, toujours plus vite, des quantités d'informations toujours plus grandes. Il n'en fallut pas plus pour que le gouvernement américain décide de débloquer d'importants crédits pour la recherche sur la traduction automatique.

On a pensé, assez "naïvement", qu'il suffirait d'introduire dans la machine le vocabulaire de la langue cible, celui de la langue source, plus un certain nombre de règles de grammaire. Dès 1960, BAR-HILLEL (4) prévoyait la faillite des réalisations pratiques en traduction automatique. Cependant, il ne fut guère écouté à cette époque.
En dépit des résultats peu satisfaisants, la collaboration entre linguistes et informaticiens, dans le domaine de la traduction automatique, ne doit pas être considérée comme un échec. En effet, elle a eu le grand mérite de poser clairement toute une série de problèmes. De plus, elle ne peut être séparée de l'histoire de l'intelligence artificielle, donc d'un domaine informatique.

L'approche syntaxique.

Les progrès réalisés ces dernières années dans la formalisation de la syntaxe des langues naturelles ont été considérables. Il semble que les grammaires transformationnelles ou grammaires génératives (5) sont capables de décrire la plupart des phénomènes syntaxiques d'une langue naturelle. Encore faudrait-il démontrer qu'elles sont capables de décrire tous les phénomènes syntaxiques et qu'elles sont adéquates à formaliser la syntaxe de plusieurs langues, or l'essentiel des résultats obtenus s'appuie principalement sur l'anglais.

La description d'un phénomène, quand bien même exhaustive et rigoureuse, ne suffit pas toujours à le simuler de manière immédiate sur ordinateur. Dans le domaine des grammaires transformationnelles, il s'agit de trouver l'automate leur correspondant, c'est-à-dire un modèle mathématique de ces grammaires. Depuis le début des travaux de W.A. WOODS (6) sur les réseaux augmentés de transition, plus connus sous leur sigle anglais ATN (Augmented Transition Network), on peut considérer, aujourd'hui, que le problème est en partie résolu.
Dès lors, on pourrait croire qu'il n'y a plus d'obstacle à la traduction automatique, mais ce serait considérer qu'une langue naturelle se constitue exclusivement d'un vocabulaire et d'un ensemble de règles de grammaire qui décrivent la manière dont les différents éléments de ce vocabulaire s'articulent les uns avec les autres.

L'approche sémantique.

Il n'est certes pas facile d'énumérer de manière précise et exhaustive tous les éléments que recouvre le langage. Du point de vue de l'informaticien, c'est-à-dire du point de vue des langages formels, comme le sont les langages de programmation, on a coutume de désigner par sémantique, tous les aspects du langage qui ne relèvent pas directement de la syntaxe. Ce qui ne signifie pas que la sémantique d'un langage de programmation n'obéit pas à des règles précises. Elle peut, en effet, toujours s'exprimer de manière formelle. Par contre, on se gardera de dire ce qu'est la sémantique dans le cadre d'une langue naturel-

le ceci d'autant plus que la théorie de la sémantique n'en est qu'à ses débuts et qu'elle englobe des concepts relativement différents, selon qui en parle.

On peut rattacher à la théorie de la sémantique les travaux de nombreux auteurs, qui proposent une approche du phénomène linguistique très différente de celle des syntacticiens et qui très souvent utilisent l'ordinateur comme principal outil. Ces travaux ont ceci en commun que tous manipulent des informations de nature sémantique et que tous représentent ces informations à l'aide de graphes, qui sont tantôt appelés: réseaux sémantiques, mémoires sémantiques ou encore réseaux systémiques.
Il faut noter que le terme de "réseau" constitue certainement un abus de langage du point de vue de la théorie des graphes selon laquelle un réseau est un graphe valué.

Y.A. WILKS (7) et R.C. SCHANK (8) proposent tous deux un système de représentation sémantique relativement voisin par le fait que les noeuds du graphe sont structurés de manière complexe et liés entre eux par un petit nombre de types d'arcs. Les mémoires sémantiques de M.R. QUILLIAN (9) adoptent une tendance inverse qui consiste à n'avoir qu'une information dans chaque noeud du graphe mais de nombreux types d'arcs les reliant. Par rapport à ces deux approches, mais sans entrer dans les détails de ce que sont les réseaux systémiques, puisqu'ils seront longuement discutés par la suite, disons simplement que leurs noeuds sont liés entre eux par un seul type d'arc. Quant aux noeuds eux-mêmes, il peuvent soit contenir une information unique, soit être constitués d'un sous-réseau et par conséquent représenter une structure complexe.

Il convient ici de faire une remarque importante: le cadre dans lequel nous utilisons les réseaux systémiques n'a aucun rapport avec les travaux des auteurs sus-mentionnés. Cependant, comme l'a montré M.A.K. HALLIDAY (10), cela ne signifie pas non plus que les réseaux systémiques sont inadéquats pour représenter des informations sémantiques.

La communication homme-machine.

Enfin, signalons un dernier domaine dans lequel la linguistique joue un rôle important, il s'agit de la communication homme-machine. Il est évident que les ordinateurs sont toujours plus complexes et demandent de la part de ceux qui les utilisent des qualifications toujours plus poussées. On en arrive à une situation paradoxale dans laquelle seul un petit nombre d'individus est capable de faire fonctionner les ordinateurs, alors que de plus en plus de gens, pour ne pas dire tout le monde, sont directement concernés par ceux-ci. Il est certain que le pouvoir de l'informatique concentré dans les mains d'un petit groupe d'individus, peut à la longue constituer un danger.

En outre, on constate la disparition ou l'évolution radicale d'un certain nombre de métiers dans lesquels apparaît l'informatique. L'effort d'adaptation de ceux qui doivent se convertir à cette science nouvelle est en général considérable, car la communication avec l'ordinateur est souvent très astreignante et très mal adaptée aux formes de communication dont la plupart des gens ont l'habitude.

Tous ces phénomènes ont poussé un certain nombre d'informaticiens à "humaniser" la communication homme-machine. Cependant, il faut bien admettre que l'état actuel des recherches n'en est encore qu'à ses débuts.
Bien sûr, il ne s'agit pas, du moins à court terme, d'imaginer une machine capable de dialoguer avec un individu de tout et de n'importe quoi. La conception d'une telle machine consisterait à dépasser de beaucoup tous les problèmes rencontrés dans la traduction automatique. Par contre, on peut imaginer des machines capables de comprendre un certain nombre de données dans une situation bien précise.

Un bon exemple d'une telle machine est le programme "SHRDLU" (11) de T. WINOGRAD. Cette machine connaît un ensemble d'objets, des cubes, des pyramides et une boîte, et est capable de les manipuler, et de donner des informations sur leur situation; la communication ayant lieu dans une langue naturelle, en l'occurrence l'anglais.

La réalisation d'une telle machine pose essentiellement trois problèmes:

- la machine doit posséder une certaine connaissance des langues naturelles. Dans le cas de SHRDLU, cette connaissance est relativement bonne. Notons à ce propos, qu'elle est réalisée grâce au langage PROGRAMMAR qui utilise des réseaux systémiques.

  Les toutes premières machines, comme ELIZA (12), n'ont pour ainsi dire pas utilisé de notions linguistiques. Celles-ci se bornent à repérer par "pattern matching" un certain nombre de mots-clés et à restituer des phrases fabriquées à l'aide de modèles pré-enregistrés, combinés avec les mots-clés repérés. Par contre, les machines plus récentes s'appuient toutes sur des concepts linguistiques comme les grammaires indépendantes de contexte, les ATN, ou encore comme on vient de le dire, les réseaux systémiques.

- la machine doit posséder une certaine connaissance du monde. Dans le cas de SHRDLU, il s'agit évidemment d'une connaissance très restreinte, puisqu'elle se limite à un très petit univers, essentiellement géométrique.

- enfin, la machine doit être capable d'effectuer un certain nombre de déductions. Comme on le verra par la suite, l'utilisation que nous faisons des réseaux systémiques offre, assez naturellement, la possibilité de réaliser en partie ce dernier aspect.

1.2 Principaux caractères de la linguistique systémique

La linguistique est l'étude du langage et des langues naturelles. Cependant, il conviendrait plutôt de parler "des linguistiques" que de "la linguistique". Le terme de langage recouvre un ensemble de réalités fort diverses. La linguistique peut donc se subdiviser en différents genres de linguistiques, selon que

l'accent est mis sur tel ou tel aspect du langage ou encore, selon les méthodes utilisées pour étudier le langage. Parmi les linguistiques on peut citer la linguistique générale, la linguistique appliquée, la sociolinguistique, etc...

La linguistique générale, que l'on appelle aussi parfois linguistique théorique, s'intéresse à la nature du langage. Elle tente de mettre en évidence ce que les langues particulières ont en commun et elle étudie la manière dont elles fonctionnent.

* La linguistique systémique est une école qui se rattache à la linguistique générale.

Depuis la Grèce antique jusqu'à fort récemment, la plupart des auteurs qui s'intéressaient au langage étaient "obsédés" par la question de correction (au sens de: "être correct"). Un énoncé est considéré comme correct lorsqu'il est non seulement conforme à la grammaire de la langue mais également aux règles du "bien-dire", fixées par une couche sociale dominante.
On avait coutume de citer des livres de grammaire pour affirmer quelles formes du langage sont correctes et quelles formes ne le sont pas. Trop souvent, ces livres ne traduisaient que les préjugés de leurs auteurs. Ce que ceux-ci considéraient comme correct, ne reflétait que certaines formes qu'ils auraient voulu voir utlisées par chacun.
Une telle approche de la linguistique est connue sous le nom d'approche prescriptive.

A l'approche prescriptive, s'oppose l'approche descriptive. Cette approche consiste à décrire un système linguistique dans son fonctionnement réel et non pas à le forcer à être ce qu'on voudrait qu'il soit ou à procéder à des restrictions arbitraires fondées sur un usage idéal.

Cette approche considère que toutes les variantes d'une langue sont également estimables et que, par conséquent, toutes valent la peine d'être décrites. Elle permet en outre, de mettre en évidence les caractéristiques de certains types de langages particuliers, comme les dialectes ou les différents registres d'une même langue, et d'étudier leur place dans d'autres

disciplines telles que la psychologie (psycholinguistique) ou la sociologie (sociolinguistique).

* La linguistique systémique adopte une approche descriptive du langage.

On peut considérer J.R. FIRTH comme le père de la linguistique systémique; le linguiste anglais M.A.K. HALLIDAY (13) et son école (14) en sont actuellement les principaux représentants.

Selon FIRTH, le sens constitue l'étude fondamentale du langage. Il s'intéressa donc, entre autres, à la théorie contextuelle du sens par laquelle il relie le langage aux autres aspects de l'activité humaine. Cette théorie prend comme point de départ les travaux de l'anthropologue B. MALINOWSKI. Celui-ci, devant étudier des documents éthnographiques polynésiens, constata les immenses difficultés qu'il y avait à traduire certains termes et certains textes, issus de la langue d'une culture donnée, dans la langue d'une autre culture. Car une langue est entièrement dépendante de la société dans laquelle elle est utilisée et évolue en fonction de cette société. Inversement, l'usage et la nature d'une langue peuvent être riches d'enseignements sur les caractéristiques de la société qui l'utilise.

La théorie contextuelle du sens décrit les fonctions de n'importe quelle unité linguistique en tenant compte des contextes internes (grammaire, lexique, phonologie) et des contextes externes (diverses circonstances extra-linguistiques; en particulier, sociologiques).

* la linguistique systémique, met principalement l'accent sur l'aspect sociologique du langage. Cette manière de procéder se distingue de la plupart des autres écoles linguistiques en ce sens qu'elle mélange la syntaxe et la sémantique. En fait, la linguistique systémique ne s'intéresse à la syntaxe qu'en tant que réalisation de choix sémantiques.

SAUSSURE faisait déjà une distinction entre ce qu'il appelait la langue et la parole. Il expliquait cette différence à l'aide

d'une analogie empruntée au domaine musical. La "langue" correspondrait à une pièce de musique tandis que la "parole" se rapporterait à une interprétation que l'on pourrait en donner. Il est clair que diverses interprétations d'une pièce de musique donnée peuvent être très différentes, quand bien même à leur base on trouve quelque chose de constant: la pièce de musique elle-même. Par analogie, la "langue" est la structure constante du langage qui se manifeste par la "parole", réalisée à un moment donné, dans une situation donnée. La parole est donc ce que l'on "fait" de la langue à un moment précis. Pour désigner ce concept, les linguistes systémiques parlent de comportement linguistique actuel, tandis que les linguistes transformationnels parlent plutôt de performance. Il s'agit de deux termes différents, mais visant à désigner un même concept. Quant à la "langue", elle est considérée par la linguistique systémique, comme un ensemble de possibilités pour faire. Cette manière de considérer la langue est un des traits dominants de la linguistique systémique, par opposition à la linguistique transformationnelle qui considère plutôt la langue comme une forme du savoir.

Pour la linguistique systémique, la "langue", qui est souvent désignée par le terme de comportement linguistique potentiel, est donc un ensemble d'options (partie constante et stable) parmi lesquelles une personne donnée peut faire un choix en fonction de ce qu'elle désire exprimer et de la culture à laquelle elle appartient.
Comme on le verra par la suite, il s'agit là d'une notion fondamentale puisqu'elle est à la base du modèle employé par la linguistique systémique, modèle qui sera utilisé pour analyser des données informelles.

En résumé, on peut considérer que :

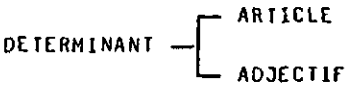
- la "langue" est un ensemble de systèmes, appelés "réseaux systémiques", qui ne sont rien d'autre qu'un graphe.

- la "parole" est un choix dans le réseau systémique, c'est-à-dire le parcours d'un certain chemin dans le graphe.

1.3 Les systèmes

La linguistique systémique tire l'origine de son nom du fait qu'elle représente le langage comme un ensemble de systèmes. Le terme de "système" est utilisé parce qu'il est le concept fondamental de la grammaire. La grammaire elle-même est basée sur la notion de choix. L'utilisateur d'une langue peut être considéré comme quelqu'un qui est amené à faire simultanément puis successivement un certain nombre de choix distincts et c'est le système qui formalise cette notion de choix dans le langage.

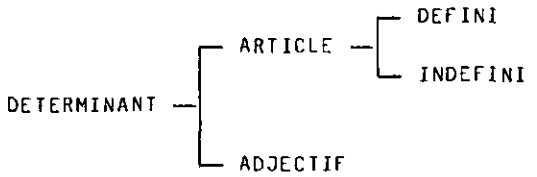
Par exemple, en grammaire française, on considère qu'un "déterminant" est soit un "article" soit un "adjectif". Un tel ensemble de choix exclusifs serait représenté de la manière suivante:



Dans cet exemple, le nombre d'options est 'deux'; plus généralement, il peut s'agir d'un nombre fini quelconque.

Les éléments d'un système sont appelés des termes ou des options et ils s'excluent mutuellement.

L'un des concepts importants, dans la construction d'un système, est celui de raffinement successif (stepwise refinement). Ce concept, que l'on désigne par le terme général de "conception top-down" consiste à aller par pas successifs du général au plus particulier.
Ainsi, on sait qu'un article peut être "défini" ou "indéfini". Dès lors le système peut être complété de la manière suivante:



Dans ce système, "article" est appelé condition d'entrée. En effet, "défini" et "indéfini" ne peuvent être choisis que si "article" a été choisi.
Donc, pour chaque système, il y a un ensemble particulier de circonstances qui doivent être satisfaites avant qu'il ne soit possible d'opérer un choix entre les termes.

La notion suivante est celle de simultanéité. Considérons les systèmes du "genre" et du "nombre" de la grammaire française. On aurait:

On sait, en grammaire française, qu'à la notion de "déterminant" est toujours liée celle du "genre" et du "nombre". En d'autres termes, lorsque l'on parle d'un "déterminant", on peut toujours, simultanément, parler de son "genre" et de son "nombre".
Dans les exemples ci-dessus, on a affaire à trois systèmes disjoints, qui tous trois sont représentés à l'aide d'une structure d'arbre. L'introduction de la simultanéité n'augmente pas seulement le pouvoir d'expression, mais étend la structure. D'un ensemble d'arbres disjoints (les systèmes) on passe à une structure plus générale, celle de graphe; c'est ce que l'on appelle un réseau systémique, c'est-à-dire un réseau (graphe) constitué de systèmes (arbres).
Pour indiquer la simultanéité, on utilise une accolade ouvrante. Dès lors, on a le réseau de la figure 1.1.

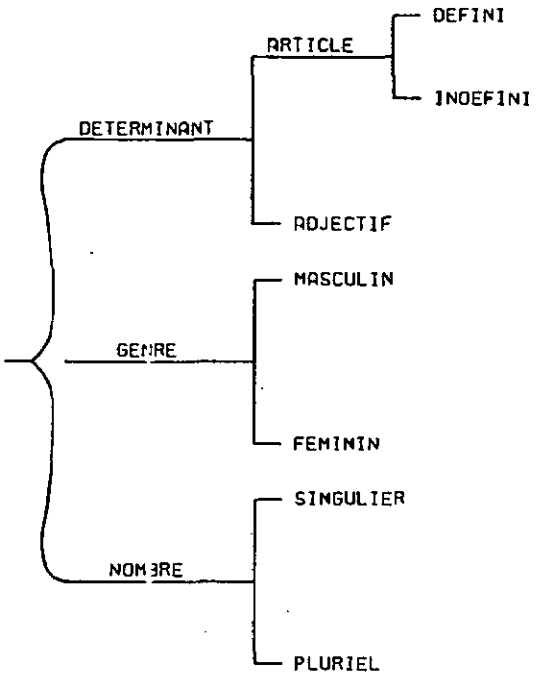


Figure 1.1

Un tel réseau doit s'interpréter de la manière suivante: après avoir choisi une des trois options possibles dans le système du "déterminant", on doit choisir une option dans le système du "genre" puis finalement, on doit choisir une option dans le système du "nombre".

L'ensemble de tous les choix possibles d'un réseau constitue ce que l'on appelle l'ensemble des paradigmes. Pour avoir un sens, un choix effectué dans un réseau donné doit nécessairement appartenir à l'ensemble des paradigmes.

Dans l'exemple ci-dessus, l'ensemble des paradigmes se constitue de douze éléments, chaque élément de l'ensemble étant constitué lui-même d'un triplet.

On sait que "le" est un article défini, masculin, singulier. Dès lors, on aimerait pouvoir exprimer que si l'on est conduit à choisir un "article défini" de genre "masculin" et dont le nombre est "singulier" on a affaire à "le". Cela est possible à condition d'introduire un concept supplémentaire qui est celui de condition et que l'on représente à l'aide d'une accolade fermante. On aboutit finalement au réseau de la figure 1.2.

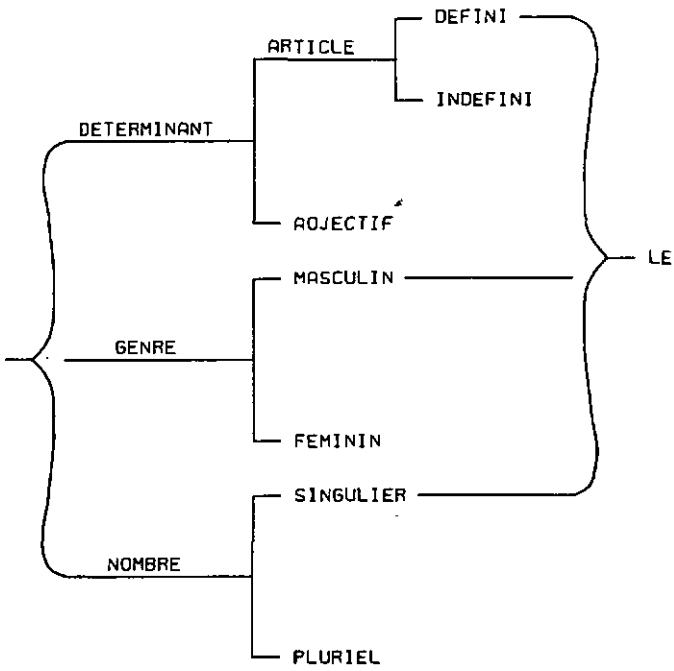


Figure 1.2

Il convient de noter ici que le concept de condition conduit à deux interprétations possibles:

- comme dans l'exemple traité, il exprime la réalisation d'un paradigme. Il permet donc de dire que l'une des réalisations possibles - en l'occurrence, la seule - du paradigme 'article défini masculin singulier' est le mot "le". Dans ce cas, il serait plus correct de parler de "réalisation" plutôt que de "condition". Cependant, par abus de langage, il ne sera pas fait de distinction terminologique.

- il permet d'exprimer une condition d'entrées multiples. Il s'agit donc d'un choix systémique. En d'autres termes, il permet de dire qu'une configuration donnée de choix conduit de manière unique à d'autres choix. Inversement, ces derniers choix ne sont accessibles que si et seulement si la configuration donnée est choisie.

Le terme de "simultanéité" fait en général référence au "temps". Il serait faux d'en conclure qu'il y a une notion temporelle dans un réseau. Un réseau ne doit pas être considéré comme un diagramme que l'on parcourt en effectuant au fur et à mesure du parcours des choix successifs, mais plutôt comme un instrument permettant des descriptions atemporelles qui appartiennent nécessairement à l'ensemble des paradigmes. Pour désigner une description, on lui associe une étiquette qui n'est rien d'autre que l'ensemble des options choisies dans le réseau. Par exemple, on a l'étiquette 'déterminant - adjectif - féminin - pluriel'.

Le réseau ne dit rien sur la forme ou la syntaxe des paradigmes qu'il spécifie, pas plus qu'il ne précise la manière de les reconnaître. Il dit simplement, dans l'exemple, qu'il y a des "déterminants" et ce qu'est un "déterminant".

Il est évident que le fait de pouvoir décrire des éléments, même de façon claire et précise, n'apporte rien en soi. Tout l'intérêt des réseaux réside dans le fait qu'ils donnent une signification à un terme en fonction de la signification des autres termes. Ainsi, par exemple, si l'on considère l'élément du lexique "animal", bien qu'il ait un sens en soi, on ne sait pas

exactement ce qu'il signifie tant que l'on n'a pas dit s'il s'agissait d'un "non (végétal ou minéral)" ou d'un "non-humain" ou encore d'un "non (oiseau ou reptile ou poisson)". La description d'un élément n'a véritablement de sens qu'en fonction des caractéristiques qu'il a (son étiquette) mais également en fonction des caractéristiques qu'il n'a pas, du niveau auquel se situe cette description et du contexte qui l'entoure, toutes choses que l'on peut considérer comme des informations sémantiques et qui apparaissent très clairement dans un réseau systémique. L'utilisation de cette information sémantique n'est pas toujours chose aisée et ne se fait, en général, que de façon empirique.

Il a été brièvement mentionné, au premier paragraphe de ce chapitre, que les réseaux offraient naturellement la possibilité d'effectuer certaines déductions. En effet, si l'on se réfère, par exemple, au terme "indéfini" de la figure 1.2, le réseau est capable de déduire, parce qu'il le "sait" de manière intrinsèque, qu'il s'agit d'un "article", lequel est un "déterminant" et qu'à cette dernière notion sont toujours liées celles du "genre" et du "nombre". A ce propos, il convient de souligner que le choix des noms qui figurent dans le réseau est purement arbitraire ou, plus exactement, que la pertinence de ce choix incombe au seul concepteur du réseau. Rien ne l'empêcherait d'utiliser le nom "monsieur" ou encore "xyz" pour "masculin". Dans ce cas, il est bien clair que les informations sémantiques et déductives, inhérentes au réseau, deviendraient non pertinentes et même parfois contradictoires.

Enfin, signalons que les relations qui existent entre les différents éléments du réseau sont presque toujours établies par rapport à des critères extérieurs à ce dernier. Par exemple, le fait de dire qu'un "déterminant" est soit un "article" soit un "adjectif" relève de critères basés uniquement sur des notions de grammaire française. Là encore, il n'existe aucun mécanisme permettant de vérifier la cohérence des relations.

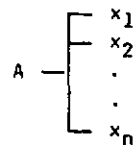
Résumé des notations:

(1) $A \rightarrow x_1$

Il y a un système x_1 dont la condition d'entrée est A.

si A alors x_1.

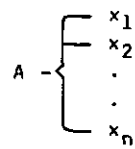
Par convention, un tel système sera appelé système simple.

(2) 

Il y a un système $x_1/x_2/.../x_n$ dont la condition d'entrée est A.

si A alors x_1 ou x_2 ou ... ou x_n ($n \geq 2$), ou exclusif.

Par convention, un tel système sera appelé système-bar, puisqu'il est noté à l'aide d'une barre.

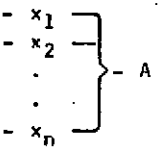
(3) 

Il y a un système simultané $x_1/x_2/.../x_n$ dont la condition d'entrée est A.

si A alors x_1 et x_2 et ... et x_n ($n \geq 2$).

Par convention, un tel système sera appelé système-bra, puisqu'il est noté à l'aide d'un "bracket" (accolade).

(4)

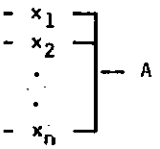


Il y a un système A dont les conditions d'entrée simultanées sont x_1, x_2, ... x_n.

<u>si</u> x_1 <u>et</u> x_2 <u>et</u> ... <u>et</u> x_n <u>alors</u> A ($n \geq 2$).

Par convention, un tel système sera appelé un <u>système-con</u>, puisqu'il indique une condition.

(5)



Il y a un système A dont les conditions d'entrée sont x_1, x_2, ... x_n.

<u>si</u> x_1 <u>ou</u> x_2 <u>ou</u> ... <u>ou</u> x_n <u>alors</u> A ($n \geq 2$).

Par convention, un tel système sera appelé un <u>système-any</u>, puisqu'il a comme conditions d'entrée un "quelconque" x_1, x_2, ...x_n.

A noter que ce système n'offre pas un grand intérêt, mais qu'il a plutôt été mis là par esprit de symétrie et de généralité.

REFERENCES

1. SAUSSURE, F. de, Cours de linguistique générale. Paris, Payot, 1916.

2. CHOMSKY, N., Three models for the deacription of language. I.R.E. Transactions on Information Theory IT, 1956, 2, 113-124.

CHOMSKY, N., Syntactic Structures. La Haye, Mouton, 1957.

CHOMSKY, N., On certain formal properties of grammars. Inf. and Control, 1959, 2, 137-167.

3. NAUR, P. (Ed.), Report on the algorithmic Language ALGOL 60. Comm. ACM, 1960, 3, 229-314.

4. BAR-HILLEL, Y., The present status of automatic translation of languages. Advances in Computers, New York, Academic Press, 1960, 1, 91-163.

5. CHOMSKY, N., Aspects of the Theory of Syntax. Cambridge (Mass.), The MIT Press, 1965.

RUWET, N., Introduction à la grammaire générative. Paris, Plon, 1967.

6. WOODS, W.A., Transition Network Grammars for Natural Language Analysis. Comm. ACM, 1974, 13, 591-606.

7. WILKS, Y.A., Grammar, meaning and the machine analysis of language. London, Routledge and Kegan Paul, 1972.

8. SCHANK, R.C., Identification of conceptualizations underlying natural language, in Schank, R.C. et Colby, K.M. (Eds), Computer Models of Thought and Language, San Francisco, Freeman, 1973.

9. QUILLIAN, M.R., Semantic Memory, in Minsky, M. (Ed.), Semantic Information Processing, Cambridge (Mass.), The MIT Press, 1968.

10. MALLIDAY, M.A.K., La Sémantique et la syntaxe dans une grammaire fonctionnelle (vers une sémantique sociologique), in Pottier, B. (Ed.), Sémantique et logique. Paris, Editions Universitaires, 1976.

11. WINOGRAD, T., Understanding Natural Language. New York, Academic Press, 1972.

BODEN, M., Artificial Intelligence and Natural Man. Massocks, The Harvester Press, 1977.

12. WEIZENBAUM, J., ELIZA - A Computer Program for the Study of Natural Language Communication Between Man and Machine. Comm. ACM, 1966, 9, 36-45.

13. MALLIDAY, M.A.K., Exploration in the Functions of Language. London, Arnold E., 1973.

HALLIDAY, M.A.K., System and Function in Language. Kress, G.R. (Ed.), Selected Papers. London, Oxford University Press, 1976.

14. BERRY, M., An Introduction to Systemic Linguistics, 1 Structures and Systems. London and Sydney, Batsford B.T., 1975.

BERRY, M., An Introduction to Systemic Linguistics, 2 Levels and Links. London, Batsford B.T., 1977.

CHAPITRE DEUXIEME

FORMALISATION DES RESEAUX SYSTEMIQUES

Ce chapitre décrit de manière "algébrique" ce qu'est un réseau systémique. Le formalisme utilisé est emprunté à la théorie des grammaires formelles. Chacun des cinq systèmes introduits en 1.3 est discuté en détail. Enfin, on montrera que tout langage indépendant de contexte peut s'exprimer à l'aide d'un réseau systémique.

2.1 Grammaires formelles

Notations et définitions:

- Un alphabet ou vocabulaire V est un ensemble fini non vide de symboles quelconques.

- Une séquence $s = s_1 s_2 \ldots s_n$, composée de symboles non nécessairement distincts et appartenant au vocabulaire V, est appelée une chaîne ou un mot (défini sur V).

- L'opération qui consiste à écrire une chaîne $v = y_1 y_2 \ldots y_n$ après une chaîne $u = x_1 x_2 \ldots x_m$ pour former la nouvelle chaîne $s = uv = x_1 x_2 \ldots x_m y_1 y_2 \ldots y_n$ s'appelle la concaténation de v à u.

- La longueur d'une chaîne s, notée LENGTH(s), est le nombre de symboles qui figurent dans la chaîne. Ici, on a LENGTH(s) = LENGTH(uv) = LENGTH(u) + LENGTH(v) = m + n.

- On note ϵ la chaîne telle que LENGTH(ϵ) = 0 et on l'appelle chaîne vide.

- V^* dénote l'ensemble de toutes les chaînes formées à l'aide des symboles de V.

- On note V^+ l'ensemble V^* auquel on enlève la chaîne vide. Donc $V^+ = V^* - \{\epsilon\}$.

- Un langage (défini sur le vocabulaire V) est un sous-ensemble, au sens large, de V^*.

Si un langage se constitue d'un ensemble fini de chaînes, il n'est pas difficile de le définir; il suffit de donner la liste de toutes les chaînes qui le composent. Par contre, si le langage est infini, on utilisera une grammaire pour en donner une représentation finie.

Formellement, une grammaire G est un quadruplet noté:
G = (N,T,P,S) où:

1) N, est un ensemble fini non vide de symboles appelé vocabulaire non-terminal et dont les éléments s'appellent non-terminaux.

2) T, est un ensemble fini non vide de symboles appelé vocabulaire terminal et dont les éléments s'appellent terminaux. T et N sont deux ensembles disjoints. On notera par V l'ensemble constitué de la réunion de N et T.
Donc V = N ∪ T et est appelé le vocabulaire.

3) P, est un ensemble fini dont les éléments s'appellent des productions ou règles de grammaire et qui consistent en des expressions de la forme:
$u \rightarrow v$, où $u \in V^+$ et $v \in V^*$.

4) S, est un symbole particulier de N, que l'on appelle axiome ou symbole start.

Par convention, on utilisera des lettres majuscules pour désigner les symboles non-terminaux; des lettres minuscules du début de l'alphabet (a,b,c,...) pour désigner des symboles terminaux et des lettres minuscules de la fin de l'alphabet (u,v,w,...) pour désigner des chaînes quelconques de terminaux ou de non-terminaux.

Pour comprendre comment une grammaire peut définir un langage, il suffit de considérer le symbole "→" de 3) comme une

sbréviation pour "peut être remplscé par". Notons, su passage, que c'est le logicien E. POST qui a imaginé cet opérateur pour décrire certains systèmes formels.
On part de l'axiome que l'on remplace par la chaîne située à droite du symbole "→" d'une production de la forme S → u. A l'side des productions de G, on remplace tout symbole non-terminal de u et l'on obtient ainsi de nouvelles chaînes. Si en appliquant récursivement ce processus, on aboutit à la chaîne vide ou à une chaîne qui ne comporte que des non-terminaux, alors u appartient su langage défini par G et on dit que u dérive de S dans la grammaire G.
Le langsge que l'on note L(G) se constitue donc de l'ensemble de toutes les chaînes que l'on peut dériver à partir de S.

Par exemple, l'ensemble des nombres binaires, qui est infini, peut se définir par la grammaire G = (N,T,P,S) avec:

N = {S}; T = {0,1}; P = {S → 0, S → 1, S → 0S, S → 1S}.

Ici, S est le seul non-terminal; il y a deux terminaux, 0 et 1 et quatre productions.
Le nombre cinq, qui en binaire s'écrit 101 sera dérivé à partir de l'axiome S de la manière suivante:

S → 1S → 10S → 101.

Classification des grammaires.

La notion de grammaire telle qu'on vient de la définir a été introduite par N. CHOMSKY (1). Il a en outre effectué une classification des grammaires selon quatre types, en imposant certaines restrictions sur la forme des productions. Ces quatre types sont:

Type 0: aucune restriction sur les productions.

Type 1: toutes les productions sont de la forme:
$u \rightarrow v$ avec LENGTH(u) ≤ LENGTH(v) et $u,v \in V^+$.
Une grammaire de type 1 est dite dépendante de contexte.

Type 2: toutes les productions sont de la forme:
$A \rightarrow v$ avec $A \in N$ et $v \in V^*$.
Une grammaire de type 2 est dite indépendante de contexte.

Type 3: toutes les productions sont de la forme:
soit $A \rightarrow a$ soit $A \rightarrow aB$ avec $a \in T$ et $A,B \in N$.
Une grammaire de type 3 est dite régulière.

La grammaire de l'exemple ci-dessus qui engendre les nombres binaires est donc de type 3.

Remarques:

1) Le langage porte le même nom que la grammaire qui l'engendre. Ainsi, un langage engendré par une grammaire indépendante de contexte sera dit langage indépendant de contexte ou de type 2.

2) Il est clair qu'une grammaire régulière satisfait les propriétés d'une grammaire indépendante de contexte, l'inverse n'étant pas nécessairement vrai. Plus généralement, toute grammaire de type (n+1) est de type n, n=0,1,2.

3) Par rapport aux définitions, la chaîne vide ϵ n'est pas autorisée dans le membre de droite d'une production appartenant à une grammaire dépendante de contexte, mais l'est dans une grammaire indépendante de contexte. Cette différence n'est pas essentielle car on peut montrer (2) que pour toute grammaire G indépendante de contexte, il y a une grammaire G' ne comportant pas de ϵ-production (une production de la forme $A \rightarrow \epsilon$) telle que $L(G) = L(G') - \{\epsilon\}$. Il est clair que G' est également indépendante de contexte. Par conséquent la remarque 2) qui dit que toute grammaire indépendante de contexte est dépendante de contexte reste vraie à condition d'enlever la chaîne vide, si elle existe, dans le langage indépendant de contexte.

4) Certains auteurs, pour marquer la récursivité, utilisent la notation: $u \rightarrow \{v\}$

Ce qui signifie que u peut se dériver en v, vv, ... un nombre fini quelconque de fois.
Il ne s'agit là que d'une question d'écriture puisque toute production de la forme:

$$u \to \{v\}$$

peut se réécrire à l'aide des deux productions:

$$u \to v$$
$$u \to uv$$

5) Enfin, on rencontre souvent la notation:

$$u \to v_1 \mid v_2 \mid \ldots \mid v_n$$

pour:
$$u \to v_1$$
$$u \to v_2$$
$$\vdots$$
$$u \to v_n.$$

2.2 Réseaux systémiques et grammaires formelles

Dans ce paragraphe, on ne considère que les systèmes 1-2-3, c'est-à-dire, le système simple, le système-bar et le système-bra introduit en 1.3.

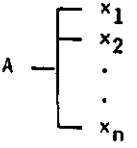
Considérons un réseau, dont la condition d'entrée est R, constitué d'une combinaison quelconque de systèmes simples, -bar ou -bra. On distingue, sur un tel réseau, deux types d'éléments; les conditions d'entrée et les termes, c'est-à-dire l'ensemble des éléments qui ne sont pas conditions d'entrée. Soit G = (N,T,P,S) une grammaire telle que N = {conditions d'entrée}, T = {termes} et S = R. Nous allons montrer comment, à partir d'un tel réseau, il est possible de construire P.

Règle 1: A tout système simple de la forme:

$$A \text{ — } x$$

on associe la production: $A \rightarrow x$.

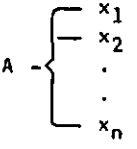
Règle 2: A tout système de la forme:



on associe l'ensemble des productions:

$$A \rightarrow x_1 \mid x_2 \mid \ldots \mid x_n.$$

Règle 3: A tout système de la forme:

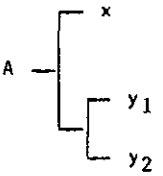


on associe la production: $A \rightarrow x_1 x_2 \ldots x_n$.

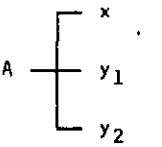
Jusque-là, aucune hypothèse n'a été formulée sur la nature des "x_i" qui figurent dans les systèmes.
Il peut s'agir d'une condition d'entrée (non-terminal) ou d'un terme (terminal), auquel cas on appliquera une des règles 1 à 3. Mais il peut également s'agir d'un système, que l'on appellera sous-système. On parlera de systèmes composés si un système comporte comme éléments un ou plusieurs sous-systèmes.

Systèmes composés.

Soit le système composé:

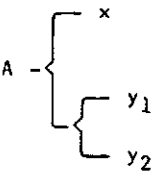


il est clair que ce sytème composé est équivalent au système:

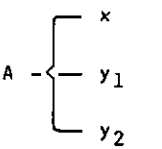


puisque si A alors x ou (y_1 ou y_2) est équivalent à si A alors x ou y_1 ou y_2.

De même, le système composé:



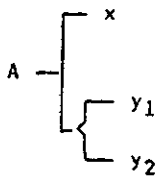
est équivalent à:



puisque si A alors x et (y_1 et y_2) est équivalent à si A alors x et y_1 et y_2.

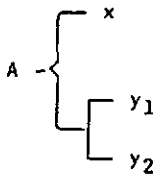
Il reste deux cas à examiner:

Règle 4: A tout système de la forme:



on associe les productions: $A \rightarrow x \mid y_1 y_2$

Règle 5: A tout système de la forme:



on associe les productions: $A \rightarrow xy_1 \mid xy_2$, puisque si A alors x et (y_1 ou y_2) est équivalent, par distributivité, à si A alors (x et y_1) ou (x et y_2).

Par mesure de simplification les règles énoncées ci-dessus l'ont été avec des systèmes qui ne se composaient que de deux éléments, mais il est clair qu'elles restent les mêmes pour des systèmes composés d'un nombre fini quelconque d'éléments. Par ailleurs, si les x_i et les y_i se composent eux-mêmes de systèmes, il suffit d'appliquer récursivement une des règles 1 à 5 jusqu'à l'obtention de toutes les productions. En outre, il est bien évident qu'on peut se passer entièrement des règles 4 et 5. Il suffit pour cela de donner, à tout sous-système, un nom de condition d'entrée arbitraire pourvu que ce nom ne figure pas déjà dans le réseau.

On a donc entièrement défini la grammaire G = (N,T,P,S) à partir

d'un réseau qui ne se constituerait que de systèmes simples, -bar ou -bra. De plus, il est clair que G est une grammaire indépendante de contexte, puisque toutes les productions sont de la forme: <condition d'entrée> → u.

Réciproquement, toute grammaire indépendante de contexte peut s'exprimer à l'aide d'un réseau systémique uniquement constitué des systèmes simples, -bar ou -bra, en appliquant les règles inverses de celles qui viennent d'être énoncées.
On notera, dans ce cas, que tout non-terminal se définit en une seule fois, c'est-à-dire qu'à tout non-terminal ne correspond qu'une et une seule définition.

2.3 Les fonctions PERE, ANCETRE et FILS

Par convention, nous appellerons R un réseau dont la condition d'entrée principale est R.

Considérons un réseau R dans lequel on a donné un nom à toute condition d'entrée de tout sous-système. De plus, on fera l'hypothèse que la condition d'entrée R n'apparaît qu'une seule fois dans le réseau. Si tel n'était pas le cas pour un réseau S, il suffirait de considérer le réseau équivalent R — S — ... (notation (1) du paragraphe 1.3) dans lequel R n'apparaît nulle part ailleurs.

La fonction PERE.

La fonction PERE associe à tout élément du réseau R, à l'exception de R lui-même, sa condition d'entrée.
En termes de grammaire on a que PERE est une fonction de V - {R} dans N.

On introduit la notation: $PERE^n(x) = PERE(PERE^{n-1}(x))$.

Pour tout élément x du réseau R, il existe un certain entier n, tel que $PERE^n(x) = R$; n sera appelé le niveau de x dans R.

On définit $PERE^*(x)$ comme l'ensemble constitué des éléments $\{PERE(x), PERE^2(x), \ldots, PERE^n(x) \mid PERE^n(x) = R\}$. Donc, le cardinal de $PERE^*(x)$ est égal au niveau de x, c'est-à-dire n.

La fonction ANCETRE.

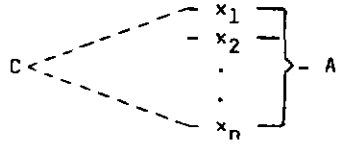
La fonction ANCETRE est une fonction à n arguments que l'on définit comme suit:

$ANCETRE(x_1,x_2,\ldots,x_n) = PERE^i(x_1) = PERE^j(x_2) = \ldots = PERE^m(x_n)$
avec i,j,...,m les puissances du premier père commun.

Donc $ANCETRE(x_1,x_2,\ldots,x_n)$ définit la condition d'entrée commune "la plus proche" à l'ensemble des éléments $x_1, x_2, \ldots, x_n$.

En particulier:

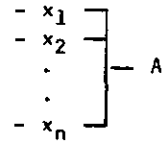
- on a que ANCETRE({éléments du réseau}) = R.
- on a que pour tout x, $PERE^*(x)$ comprend au moins R.
- dans le cas d'un système-con:



on a que : $C = ANCETRE(x_1,x_2,\ldots,x_n)$.
Comme C est la condition d'entrée unique de A, on définit:

$PERE(A) = ANCETRE(x_1,x_2,\ldots,x_n)$.

- dans le cas d'un système-any:



Comme A a plusieurs conditions d'entrée, on a que:
PERE(A) = x_1 ou x_2 ou ... ou x_n.

La fonction FILS.

La fonction FILS associe à toute condition d'entrée A, à l'exception des conditions d'entrée appartenant à un syatème-con, l'ensemble de tous les éléments directement accesaibles à pa ti e A.

En termes de grammaire on a que FILS est une fonction de:

N - {x | x est une condition d'entrée d'un système-con} dans $\{y_i \mid y_i \in V\}$.

Soit FILS(x) = $\{x_1, x_2, \ldots, x_n\}$, à tout x_i qui est une condition d'entrée n'appartenant pas à un système-con, on peut associer un nouvel ensemble $FILS(x_i)$. On obtient alors une famille d'ensembles dont on peut considérer la réunion F. On peut ensuite, appliquer la fonction FILS à tout élément de F pour lequel la fonction FILS est bien définie et ceci jusqu'à l'obtention d'un ensemble stable que l'on note $FILS^*(x)$.

Donc $FILS^*(x) = FILS(x) \cup_i FILS(x_i \in FILS^*(x)$ pour lesquels $FILS(x_i)$ est définie).

Exemple:

Considérons le réseau abstrait de la figure 2.1; par réseau abstrait, on entend, un réseau dont les éléments sont des identificateurs arbitraires qui n'ont aucune signification intrinsèque.

Dans cet exemple, l'élément B apparaît deux fois, ainsi que l'élément S. Pour désigner un élément de manière non-ambiguë, on conviendra de l'indicer à l'aide de son père, ou de l'un de ses pères dans le cas d'un système-any. Cette convention ne soulève pas toutes les ambiguïtés, auquel cas on pourra toujours parler d'un élément en le désignant par une flèche ou à l'aide du "doigt".

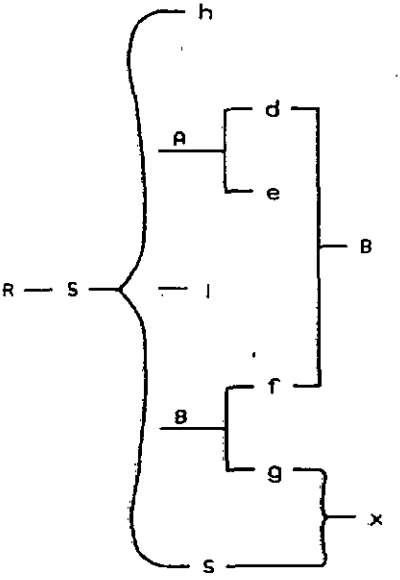


Figure 2.1

On a, entre autres:

$PERE(A) = S_R$.
$PERE(B_d) =$ d ou f.
$PERE(X) = ANCETRE(g,S_S) = S_R$.
$PERE^*(f) = \{B,S,R\}$.
$FILS(B_S) = \{f,g,\}$.
$FILS^*(A) = \{d,e,B_d\}$.
$FILS^*(S_R) = \{h,A,d,e,B,i,f,g,S\}$.

On notera que la fonction FILS n'est pas toujours la fonction réciproque de ANCETRE. En effet,dans le cas de figure 2.1 on a que $FILS(B_S) = \{f,g,B_d\}$ alors que $ANCETRE(f,g,B_d) = S_R$.

Définition:

- On dira d'un réseau R qu'il est récursif, si R $\in$ FILS*(R). Si *on a n occurrences de R, alors on dira que* R est *n-récursif*.

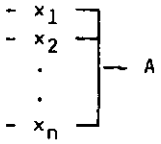
- On dira d'un système S qu'il est récursif si l'on a un des deux cas suivants:
  1) S $\in$ FILS*(S).
  2) il existe un x $\in$ FILS*(S) tel que x $\in$ PERE*(S).
  On parlera également d'un système n-récursif.

2.4 Le système-any

On va montrer comment définir en un ensemble de productions tout *système-any apparaissant dans un réseau* R. *On suppose qu'on* a donné un nom à toute condition d'entrée de tout sous-système de R.

On obtient un ensemble de productions en appliquant les règles 1 à 3 définies en 2.2 à tous les systèmes simples, -bar ou -bre du réseau.

Supposons qu'on ait un système-any:



Considérons le cas d'un x quelconque, disons x_i. Il y a dans l'ensemble des productions, une ou plusieurs productions de la forme:

$$PERE(x_i) \rightarrow ux_iv,\ u \text{ et } v \in V^*.$$

Il est clair que, dans le réseau, le système-any désigne un x_i de manière unique. Au cas où on aurait plusieurs productions de la forme ci-dessus, il faudrait choisir celle qui contient le x_i désigné par le système-any.
Dès lors, il suffit d'introduire dans l'ensemble des productions la règle:

$$PERE(x_i) \rightarrow ux_iAv$$

et ceci pour tout i = 1,2,...,n.

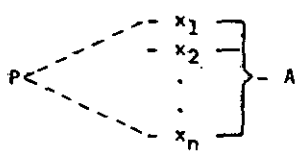
Donc pour tout système-any à n arguments, l'ensemble des productions sera augmenté de n éléments tous construits de la manière décrite ci-dessus.

Il est important de noter que l'introduction du système-any maintient le type de la grammaire à savoir, indépendante de contexte.

2.5 Le système-con

Le cas du système-con est considérablement plus complexe que que celui du système-any. Disons d'emblée qu'il n'y a pas d'algorithme général permettant de transformer n'importe quel système-con en une ou plusieurs productions.

Soit le système-con:



alors PERE(A) = ANCETRE(x_1,x_2,...,x_n) = P.

Deux cas sont à envisager:

- P est la condition d'entrée d'un système-bar.
- P est la condition d'entrée d'un système-bra.

Examinons d'abord le cas où P est la condition d'entrée d'un système-bar. Ce cas peut sembler paradoxal puisque, par définition, on doit opérer un choix exclusif sur les éléments d'un système-bar et qu'il semble par conséquent impossible de réunir simultanément les conditions x_1 et x_2 et ... et x_n.
Cependant, il se peut qu'on ait affaire à un système-bar, de condition d'entrée P, récursif. Le seul fait que P soit récursif ne suffit pas à réaliser la condition d'entrée simultanée de A. Si A a comme condition d'entrée simultanée n éléments, il faut pour qu'elle soit satisfaite, que l'une au moins des deux conditions suivantes soient réalisées:

a) P est au moins n-récursif
b) Il existe (n-1) systèmes y_i récursifs tels que $x_j \in FILS^*(y_i)$

Si tel n'est pas le cas, la condition d'entrée simultanée de A ne sera jamais satisfaite et il est par conséquent inutile de chercher à transformer le système-con en une production ! Par contre, si l'une des deux conditions mentionnées ci-dessus est satisfaite, il n'existe malheureusement pas d'algorithme permettant de transformer un système-con en une ou plusieurs productions.

Dans un certain nombre de cas simples, il est possible de transformer tout le réseau en un réseau équivalent, dans lequel le système-con n'apparaît plus. Par exemple, on se convaincra assez aisément que le réseau de la figure 2.2 est équivalent à celui de la figure 2.3.

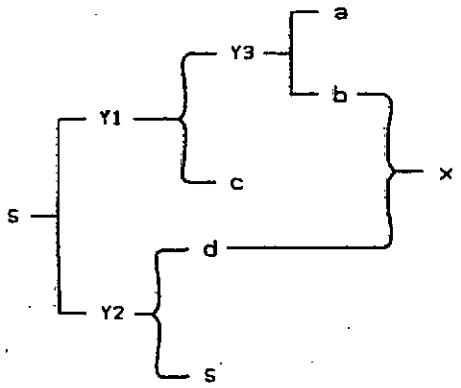


Figure 2.2

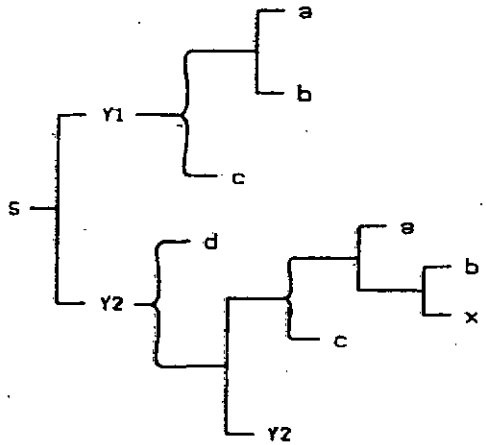


Figure 2.3

On noters que le réaesu de la figure 2.2 a les propriétéa aulvantes:

- le système-con s deux conditions d'entrée simultanées.

- PERE(x) = ANCETRE(b,d) = S.

- S est 1-récursif, il ne aatisfait donc pas la condition a). Par contre, n = 2 et il existe un système récursif y_i tel que $x_j \in FILS^*(y_i)$. Ce système eat y_2 puisque $d \in FILS^*(y_2) = \{d,S\}$.

Il nous reste à examiner le css d'un système-con, dont l'sncêtre P eat la condition d'entrée d'un aystème-brs. On envisagera les deux cas suivants:

- P n'est pas récursif.
- P est récuraif.

Supposons qu'on ait le achéma auivant:

avec $n \leq m$ et la production $P \rightarrow y_1y_2\cdots y_m$.

Soit y_k tel que $x_n \in FILS(y_k)$. Si y_k est la condition d'entrée d'un aystème-brs alors on s une production de la forme

$$y_k \rightarrow ux_nv, \; u \text{ et } v \in V^*.$$

Si P est non récursif alors on introduira une production unique de la forme:

- $P \rightarrow y_1y_2\cdots y_kA\cdots y_m$ ai y_k eat la condition d'entrée d'un ayatème-bar

- $P \rightarrow y_1 y_2 \ldots u x_n A v \ldots y_m$ si y_k est la condition d'entrée d'un système-bra

dans lequelle tant que $x_i \in FILS^*(y_j)$ on remplace y_j par:

- x_i si $PERE(x_i)$ est une condition d'entrée d'un système-bar

- la concaténation des éléments de $FILS(PERE(x_i))$ pris dans l'ordre de leur apparition dans le réseau, c'est-à-dire de haut en bas, sinon.

Dans l'exemple de la figure 2.4, on introduira la production P → bcefgAh. Partant de $P \rightarrow y_1 y_2 y_3$ on remplace: y_1 par bc puisque PERE(c) = BC et que FILS(BC) = {b,c}; y_2 par e puisque PERE(e) = y_2, condition d'entrée d'un système-bar; y_3 par fgAh puisque y_3 est de la forme $y_3 \rightarrow fgh$.

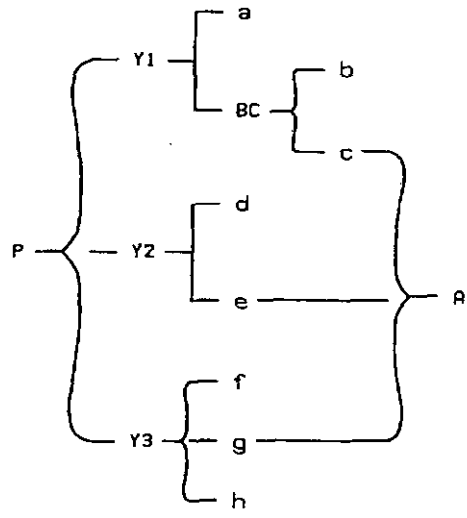


Figure 2.4

Finalement, si P (ancêtre d'un système-con) est la condition d'entrée d'un système-bra récursif, on est ramené au même cas que celui où P est la condition d'entrée d'un système-bar. A la différence près que la récursivité n'intervient pas dans la réalisation des conditions d'entrée simultanées du système-con. En effet, quelque soit le degré de récursivité, les conditions d'entrée simultanées pourront être satisfaites puisqu'il s'agit d'un système-bra.
De même qu'il n'y a pas d'algorithme permettant de définir le système-con à l'aide de productions si P est la condition d'entrée d'un système-bar, de même, il n'y a pas d'algorithme permettant de définir le système-con à l'aide de productions si P est la condition d'entrée d'un système-bra récursif.

Dans certains cas simples, il est également possible de transformer un réseau possédant un système-con en un réseau équivalent qui ne l'a plus.

En conclusion de ce chapitre, on peut donc dire que toute grammaire indépendante de contexte peut être représentée par un réseau systémique. Lorsqu'on a affaire à de petites grammaires, l'avantage de cette représentation est incontestable, puisqu'elle permet de visualiser globalement l'ensemble des relations existant entre les différents éléments. Inversement, tout réseau systémique qui ne se constitue que d'une combinaison arbitraire de systèmes simples, -bar, -bra, -any ou -con non récursifs peut toujours s'exprimer à l'aide d'une grammaire indépendante de contexte.

REFERENCES

1. CHOMSKY, N., Three models for the description of language. I.R.E. Transactions on Information Theory IT, 1956, 2, 113-124.

CHOMSKY, N., On certain formal properties of grammars. Inf. and Control, 1959, 2, 137-167.

2. BACKHOUSE, R.C., Syntax of Programming Languages: Theory and Practice. London, Prentice-Hall, 1979.

GROSS, M. et LENTIN, A., Notions sur les Grammaires formelles. Paris, Gauthier-Villars, 1967.

HOPCROFT, J.E. and ULLMANN, J.D., Formal Languages and their relation to automata. Reading (Mass.), Addison-Wesley, 1969.

CHAPITRE TROISIEME

RESEAUX SYSTEMIQUES ET ANALYSE DE DONNEES

3.1 Méthode générale

J. WEIZENBAUM a souligné avec raison et souvent avec humour ce qu'il appelle le "pouvoir de l'ordinateur" (1). Il est incontestable qu'on assiste depuis un certain nombre d'années à un besoin toujours plus grand de formalisation. Ce phénomène, qui a toujours existé dans les disciplines scientifiques et les sciences de l'ingénieur, s'étend à tous les domaines, même à ceux des sciences humaines. La démarche qui consiste à formaliser dans un souci de plus grande objectivité est certes louable mais débouche souvent sur des abus, particulièrement lorsqu'il s'agit d'utiliser l'ordinateur.

Il est une tendance malheureusement trop répandue qui consiste à considérer un résultat obtenu à l'aide d'un ordinateur comme incontestable et ceci uniquement parce qu'il a été produit par une machine. Alors qu'il est un fait connu de tout informaticien que la plupart des programmes sont imparfaits et qu'ils ne peuvent, par conséquent, que fournir des résultats souvent incorrects.

Il ne s'agit pas, bien sûr, de tomber dans l'excès contraire qui consisterait à affirmer que tout résultat obtenu par ordinateur est nécessairement incorrect. Il s'agit plutôt de prendre conscience du fait que pour pouvoir garantir une certaine validité à des résultats, il est indispensable de connaître le mieux possible la manière dont les programmes qui les calculent ont été écrits et comment utiliser précisément ces programmes.

Enfin, l'"informatisation" d'une recherche pose un certain nombre de problèmes qu'on est loin de dominer. S'il est vrai que certains domaines scientifiques comme la biologie, la géologie ou la physique atomique se prêtent particulièrement bien à des méthodes d'analyse statistique que l'on peut développer sur ordinateurs, il n'en va pas de même pour certains travaux que l'on rencontre en sciences humaines, par exemple, en psychologie, en sociologie ou en littérature.

Les raisons de ces difficultés sont évidemment multiples; on en retiendra essentiellement deux. Premièrement, la formation des gens issus des sciences humaines ne leur permet pas toujours de manipuler aisément des formalismes qui sont tous issus des mathématiques. En particulier, dans le domaine de la statistique, on sait qu'il est à peu près impossible de procéder à une analyse convenable, sans être en même temps statisticien et un spécialiste de la branche dans laquelle on procède à l'analyse. Deuxièmement, la nature et la complexité des problèmes qu'étudient les sciences humaines ne se prêtent pas facilement à la formalisation. Ceci d'autant plus qu'il n'existe pas de formalisme universel permettant de rendre compte de n'importe quel phénomène.

La plupart des chercheurs en sciences humaines doivent expliquer leur monde à l'aide de catégories. Il est clair que ce monde est infiniment complexe et que, par conséquent, les catégories le sont. La méthode présentée ici, ou plutôt le programme qui l'accompagne, propose un langage artificiel permettant de décrire des catégories de complexité arbitraire.
Cette méthode n'a évidemment pas la prétention d'offrir un "outil" universel de formalisation et d'analyse. Elle se situe dans un contexte bien précis qui est celui de l'analyse de données informelles.

Par données informelles, on entend toute donnée qui n'est pas directement issue d'un comptage ou qui peut se décrire à l'aide d'équations mathématiques, comme l'évolution d'une colonie de bactéries, dans le domaine de la biologie, ou la trajectoire d'une particule élémentaire, dans le domaine de la physique nucléaire. Les données que l'on pourra traiter et que l'on appelera, par convention, corpus, sont essentiellement des données textuelles comme des interviews, des résultats d'enquête, des questions d'examen, etc...

La méthode qui consiste à utiliser des réseaux pour représenter des données informelles n'est pas vraiment nouvelle. G. TURNER (2) a déjà utilisé des réseaux systémiques dans le cadre d'une étude sociologique dans laquelle il tente de mettre en évidence la manière dont les parents s'adressent à leurs enfants et ceci

en fonction du milieu social auquels ils appartiennent. Dans un autre domaine, J. OGBORN a utilisé les réseaux systémiques pour catégoriser les différences qui apparaissent dans des questions posées à leurs étudiants par des enseignants en physique (3).

On peut considérer que la méthode telle qu'elle est décrite ici, tire ses origines d'un travail expérimental mené par le Centre des Sciences de l'Education du Chelsea College (Université de Londres) (4) et qu'elle a ensuite été précisée par J. BLISS et J. OGBORN (5). Ce travail expérimental a consisté à interviewer 115 étudiants en physique, issus d'une dizaine d'universités différentes. Les étudiants ont été interrogés sur ce qu'ils pensaient de l'enseignement qu'ils avaient reçu et plus particullèrement sur ce qu'ils considéraient comme "bon" et ce qu'ils considéraient comme "mauvais". Le résultat des interviews a donné quelque 300 histoires informelles, matériel d'une complexité énorme qui ne se prêtait guère à des méthodes d'analyse traditionnelle. Les analystes se rendirent compte des innombrables difficultés qu'il y avait à vouloir classer tel ou tel interview ou telle ou telle partie d'interview dans une catégorie bien définie. Il fut donc décidé d'utiliser des réseaux systémiques qui permettent de définir des catégories avec de très nombreuses nuances et selon la méthode "top-down", de considérer avant tout l'essentiel, afin d'éviter de se perdre dans les détails. Par ailleurs, le simple fait de classer des interviews ou des parties d'interviews selon différentes catégories ne permet pas de rendre compte des liens et des relations existant entre ces catégories ce que permet, par contre, le concept des réseaux systémiques.

La méthode générale d'analyse à l'aide de réseaux systémiques peut se diviser en quatre étapes:
- Conception d'un réseau.

- Codification du corpus.

- Vérification de la cohérence du code par rapport au réseau et réalisation de ce code dans le réseau.

- Utilisation de l'ensemble réseau-code à des fins d'analyse.

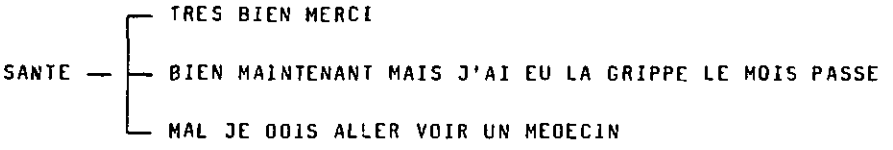
Conception du réseau.

Il est bien évident que la phase la plus délicate de la méthode réside dans la conception du réseau. Il n'y a pas de méthode générale pour cela, il s'agit essentiellement d'une démarche heuristique et empirique.

On pourrait, bien sûr, partir du corpus lui-même et tenter de développer un réseau qui en rende compte intégralement, de manière linéaire. Par exemple, si à la question "comment allez-vous ?" on obtient les trois réponses:

A) "très bien, merci"
B) "bien maintenant, mais j'ai eu la grippe le mois passé"
C) "mal, je dois aller voir un médecin"

on pourrait imaginer le réseau suivant:



Bien entendu, un tel réseau n'offre rigoureusement aucun intérêt. Il ne permet même pas de savoir combien de personnes ont répondu qu'elles allaient bien et combien ont répondu qu'elles allaient mal.

Même en admettant qu'on arrive à concevoir un réseau convenable - dans le cas de l'exemple, un réseau qui permettrait au moins de répondre à la question il y a "n personnes qui se portent bien" et "m personnes qui ne se portent pas bien" - la démarche qui consiste à construire le réseau uniquement à partir du corpus paraît dangereuse. En effet, elle a tendance à influencer l'analyste sur la manière de le concevoir, en le poussant à tenir compte d'éléments qui n'ont aucun rapport avec l'analyse ou à mettre faussement en relation des systèmes tout à fait indépendants.

Dans une première phase, la conception d'un réseau devrait être indépendante du corpus. Avant de tenir compte du corpus, l'analyste devra d'abord dégager les éléments auxquels il s'intéresse et mettre en évidence les relations qui existent entre eux. Ensuite, le processus consistera en un va-et-vient entre le corpus et le réseau afin de vérifier la validité de ce dernier. On complétera et affinera le réseau à l'aide de tests effectués sur des extraits de code, partant du principe qu'il est, a priori, impossible de tenir compte de tout.

En adoptant une telle démarche, on constatera souvent des incohérences au niveau du corpus, incohérences que l'on évitera, bien entendu, de reporter dans le réseau.

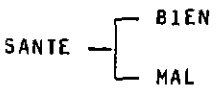
L'ensemble des éléments utilés à l'analyse et les différents liens qui les unissent constituent le réseau.

Codification.

Cette étape consiste à réécrire le corpus en utilisant les termes du réseau. Il est clair que cette réécriture pose un certain nombre de problèmes d'interprétation. Les phrases apparaissant dans le corpus ne seront que très rarement constituées de termes appartenant au réseau. L'analyste doit donc décider quelle signification donner à telle ou telle unité afin de l'exprimer à l'aide d'un ensemble de termes du réseau. Dans le cas d'unités de corpus isolées, cette décision n'est pas toujours facile, elle peut même être impossible. Par contre, il est relativement aisé, quitte parfois à prendre des décisions arbitraires, de transcrire les différents éléments en des termes du réseau, lorsqu'on a sous les yeux l'ensemble du contexte dans lequel ils se situent.

Une autre difficulté réside dans la perte d'information qu'il y a, lorsque l'on passe du corpus à sa codification.

Il y a perte d'information lorsque le réseau ne prévoit pas une situation apparaissant dans le corpus. Si, à partir de l'exemple ci-dessus, on avait le réseau:



on perdrait de l'information dans le cas de la réponse de B. En effet, la réponse de B apporte deux élements d'information: "comment B se porte maintenant" et "comment il se portait avant". Dès lors, l'analyste peut avoir deux attitudes. Soit considérer que la santé de celui qui parle ne l'intéresse qu'au moment où il pose la question, dans ce cas "j'ai eu la grippe le mois passé" ne l'intéresse pas et il laissera purement et simplement tomber les renseignements apportés par cette phrase, soit affiner le réseau pour tenir compte de cette information. Dans ce dernier cas, on aurait un réseau qui pourrait avoir l'allure de celui de la figure 3.1.

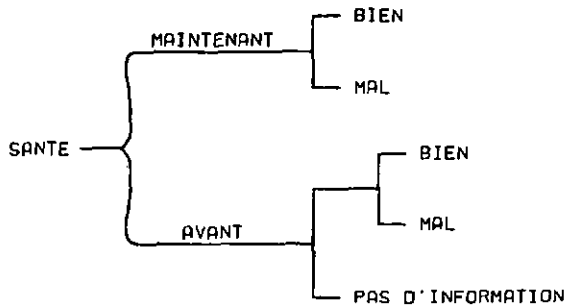


Figure 3.1

Ce point souligne d'emblée un aspect intéressant de la méthode, c'est qu'elle pousse l'analyste à poser clairement ses questions et à faire préciser, lorsque c'est possible, les réponses fournies par celui qui répond.

Enfin, doivent être mises sur le compte de la perte d'information, les renseignements que l'on considérera comme des parasites et dont il est souhaitable de ne pas tenir compte. Si à la

queation "comment allez-vous ?" on obtient la réponae "midi trente", il psrît importsnt d'élimlner une telle réponae puisqu'elle n's rien à voir svec le aystème de la ssnté.

Aprèa svoir élaboré un réaesu cohérent et résolu lea difficultés lllustréea à l'side des exemples ci-dessus, on pourra psaser à une codificstion formelle du corpua. Dsns le caa de l'exemple, sppliqué au résesu de la figure 3.1 on aurait:

A) SANTE(MAINTENANT(bien)
AVANT(paa d'lnformstion))

B) SANTE(MAINTENANT(bien)
AVANT(msl))

C) SANTE(MAINTENANT(mal)
AVANT(psa d'information))

A noter que les parenthèaes et ls structurstion en lignes n'ont sucune signification intrinsèque, elle ne sont utilisées que dans le but de restituer su mieux la structure du résesu.

Vérification et réalisstion du code.

Psr vérificstion du code, on entend: vérifier que le code, trsnacrlt de msnière formelle, appsrtient bien à l'ensemble des psradigmes du résesu, c'est-à-dire est un des chemins possibles dsns le graphe.

Cette vérificstlon est évidemment totalement mécenique, elle pourra donc être faite automstiquement sur ordinsteur et c'est préclsément l'une des choses que réslise le progrsmme BARBARA, expliqué en détsll dsns les chapitres 4 et 5.

Ls réalisstion du code dsns le réseau consiste à mémoriser sur le résesu lui-même les différents chemins empruntés psr le corpus.

Dana le css de l'exemple ci-dessua, il fsudra se souvenir que "bien" s eu comma réponse A et B tandis que "msl" n'a eu que C.

Là encore, il s'agit d'une étape purement mécanique qui sera également réalisée par le programme BARBARA.

Analyse à partir de l'ensemble réseau-code.

Cette étape consiste à interroger l'ensemble réseau-code en se posant des questions constituées à l'aide d'éléments du réseau, combinés par des opérateurs logiques.

On pourrait se poser des questions du genre:

- combien y a-t-il de réponses qui ont telle ou telle caractéristique?

- quelles sont les réponses qui ont fourni telle caractéristique mais pas telle autre?

  etc...

Là aussi, il s'agit d'une étape purement mécanique qui sera réalisée par le programme BARBARA.

Les exemples qui ont été donnés ci-dessus ne l'ont été qu'à titre purement illustratif. Ils sont évidemment beaucoup trop sommaires et simplistes pour mettre en évidence toutes les caractéristiques de la méthode d'analyse, basée sur les réseaux systémiques. Il faut toujours garder à l'esprit, qu'en général, la méthode d'analyse décrite s'applique à des données de taille et de complexité assez considérables, un réseau pouvant contenir plusieurs dizaines d'éléments et le corpus pouvant lui-même être constitué de plusieurs centaines d'histoires.

3.2 Conventions et notations

La méthode d'analyse, on l'a vu, consiste donc, dans une première phase, à construire un réseau systémique qui représentera la structure des données informelles, c'est-à-dire ce que nous sommes convenus d'appeler le corpus.

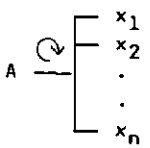
Pour construire le résesu, on conviendra d'utiliser des combinaisons quelconques des systèmes tels que les ont définia les linguistes et que nous svons appelés: aystèmes aimples, -bar, -bre, -con et -sny. Nous ne ferons aucune restriction sur la manière de combiner ces systèmes (utilisetion de systèmes composés) ni sur le nombre de leurs arguments ou sur la nature de ces arguments (termes ou conditions d'entrée).

Cependant, il convient de faire les quatre remsrques suivsntes:

1) Dn a vu en 2.3 ce qu'était un système récursif. Un tel système soulève une difficulté relative à ls lisibilité. En effet, dans un réseau d'une certaine complexité, il est extrêmement dlfficile de détecter les systèmes récursifs. La marque de la récurslvité n'est due qu'à un phénomène de synonymie, or la détection de ces synonymes oblige à garder présents à l'esprit le nom et la signification de tous les termes utilisés, ceci afin d'éviter de construire involontslrement des systèmes récuraifs.

   Aussi, conviendra-t-on d'introduire une notation supplémentaire, une boucle ↻, qui sera aux réseaux systémiques ce que la notation { } (vue en 2.1, remarque 4) est aux grammaires formelles. Un tel syatème sera appelé système-rec, abréviation de récursivité.

   A noter que la récurslvité n'a vraiment de sens que pour un système-bar. Dans ce css, un système-bar récuraif du genre:



   aura la signification suivante:

   si A alors x_1 ou x_2 ... ou x_n, ($n \geq 2$), ou non exclusif, puisque si l'on a la conditlon d'entrée A, on peut choisir un ou plusieurs x_i, i=1,2,...,n.

Quoiqu'on ne s'interdise pas de marquer la récursivité telle qu'on l'a définie en 2.3, on s'efforcera, partout où cela est possible, d'utiliser le symbole ↻.

2) On a vu, également en 2.3, les difficultés que soulèvent les synonymes lorsqu'il s'agit de les désigner. On évitera donc d'utiliser des synonymes (le cas échéant, on s'efforcera de leur donner des noms distincts de condition d'entrée), à moins qu'il ne soit indispensable de se référer plusieurs fois exactement aux mêmes termes (même concept), ce qui, en principe, n'arrive que très rarement dans les réseaux que nous utiliserons.

   A noter que deux conditions d'entrée distinctes ne peuvent jamais porter le même nom, puisque, précisément, elles définissent deux systèmes différents.

3) Du point de vue des notations, on conviendra, lorsqu'on se réfère à un terme ou à plusieurs, de les mettre entre parenthèses, en général précédés de leur condition d'entrée. Cette remarque s'applique plus particulièrement à la phase de codification. Il ne s'agit pas là d'une obligation, mais plutôt d'une recommandation visant à augmenter la lisibilité.

4) Finalement, on se donnera la possibilité d'insérer n'importe où des commentaires. Un commentaire est une séquence quelconque de caractères précédée du symbole "(*" et terminée par le symbole "*)". Où qu'il apparaisse, un commentaire doit être considéré comme ne modifiant en rien la signification du texte où il se trouve. En d'autres termes, deux textes, qu'il s'agisse d'un réseau ou de code, sont syntaxiquement identiques s'ils ne diffèrent que par leurs commentaires.

   Généralement, les commentaires apparaîtront plutôt dans le code, pour permettre de garder des éléments d'information figurant dans le corpus mais qui n'ont pas été prévus dans le réseau, c'est-à-dire auxquels ne correspond aucun code.

3.3 Traitement d'un exemple

Dans ce paragraphe, nous allons traiter un exemple complet. Il ne peut s'agir évidemment que d'un exemple relativement limité qui n'est donné qu'à titre purement illustratif.

L'expérience a consisté à poser la question suivante à une quinzaine de personnes:

"Fumez-vous? Si oui, à quelle fréquence? Dans tous les cas, dites pourquoi vous fumez, respectivement pourquoi vous ne fumez pas".

Les gens ont été priés de répondre par écrit, leur réponse ne devant pas dépasser une dizaine de lignes.

La question.

Le réseau de la figure 3.2 donnera une représentation fidèle de la question, telle qu'elle a été posée. On notera que le système "raison" est récursif. En effet, on s'attend à ce que chaque personne interrogée invoque au moins une raison (sans quoi elle n'aurait pas répondu à la question) et en général plusieurs.

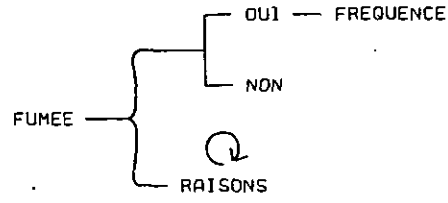


Figure 3.2

En regardant de plus près la question, on constate d'emblée qu'elle est terriblement imprécise. En effet, elle ne fait aucune allusion au "temps". Lorsqu'on demande à quelqu'un "fumez-vous?" on pourrait s'attendre à ce qu'il réponde "oui" ou "non" selon qu'il est en train de fumer ou non au moment où la question lui est posée. Il semble, probablement par convention, que tout le monde interprète la question "fumez-vous?" de la même façon, à savoir: "d'une manière générale, dans la vie quotidienne, vous arrive-t-il de fumer?", du moins c'est l'hypothèse que nous ferons.

Par contre, il peut y avoir des gens qui ne fument plus au moment où la question est posée, mais qui ont été de grands fumeurs à une période quelconque de leur vie. Comme on va principalement s'intéresser aux raisons qui poussent les gens à fumer ou à ne pas fumer, on englobera également dans ces raisons, celles qui ont poussé les gens à arrêter de fumer.

On affinera donc le réseau de la figure 3.2 pour obtenir celui de la figure 3.3. Dans ce réseau, "autrefois" doit être interprêté comme "a arrêté de fumer".

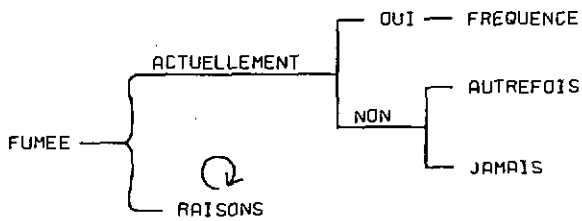


Figure 3.3

La question est également imprécise en ce qui concerne le mode de fumer (pipe, cigarette, cigare, etc...) et le genre de fumée (tabac, hachisch, marijuana, etc...). On décidera, arbitrairement, de ne pas s'intéresser au mode de fumer, quant au genre de fumée, on fera l'hypothèse qu'il ne s'agit que de tabac; la

question ayant été posée à un groupe d'étudiants et d'enseignants, dont les moeurs ne peuvent être qu'au-dessus de tout soupçon ! Il est bien clair que l'on pourrait tenir compte de tous ces éléments, il suffirait de développer à la place du système "fréquence" un système plus complet.

Enfin, le système "fréquence" ne comprendra que deux catégories: "régulier" et "occasionnel". Il s'agit là aussi de simplifier l'exemple qui, on le répète, n'est donné qu'à titre purement illustratif. On pourrait, bien sûr, imaginer un système "fréquence" beaucoup plus nuancé, pouvant aller jusqu'à tenir compte des moments de la journée auquels on fume.

Les objectifs de l'analyse.

L'objectif de l'analyse va consister à s'intéresser aux raisons qui poussent les gens à fumer ou à ne pas fumer. Ces raisons, a priori, sont certainement multiples et variées. Cependant, afin de garder un exemple aussi simple que possible, nous n'en retiendrons que trois : les raisons de nature sociale, les raisons d'ordre financier et les raisons relatives à la santé. Ces trois catégories de raisons nous conduisent à développer trois sytèmes appelés: "société", "argent" et "santé".

Toute raison qui n'est pas directement en rapport avec l'un de ces trois systèmes entrera dans un quatrième système, tout à fait général, que l'on désignera par "autre". Afin de ne perdre aucune information et de permettre ainsi une analyse ultérieure plus détaillée, on conviendra de garder "en clair", c'est-à-dire telle qu'elle apparaît dans le corpus, toute raison appartenant au système "autre".

Le système "société" définira essentiellement le rôle que les gens (famille, amis, etc...) jouent ou ont pu jouer dans les motivations qui poussent quelqu'un à fumer ou à ne pas fumer. Ce rôle sera symbolisé par un sous-système que l'on désignera par le terme de "influence". Il permettra de réaliser des notions du genre: "imiter les autres", "faire comme tout le monde", etc... Toute raison invoquée qui serait en rapport avec la société comme "c'est une question de mode", "pour se donner une

contenance en société", etc... mais qui ne relève pas du sous-système "influence", entrers dans un second sous-système de "société", désigné par le terme général de "other". Le système société est représenté dans la figure 3.4. Ce système, tel qu'il apparaît dans la figure, est évidemment considérablement simplifié. On pourrait imaginer un système beaucoup plus détaillé, si l'analyse tentait de mettre en évidence certains aspects sociologiques.

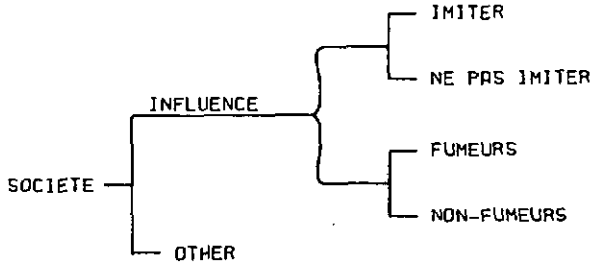


Figure 3.4

Le système "argent" sera lui aussi exprimé très simplement à l'aide de deux notions: le "coût", "cela coûte cher", "c'est bon marché", etc... et le "placement", "c'est du gaspillage", "argent jeté par les fenêtres", etc...
On aura donc le système de la figure 3.5.

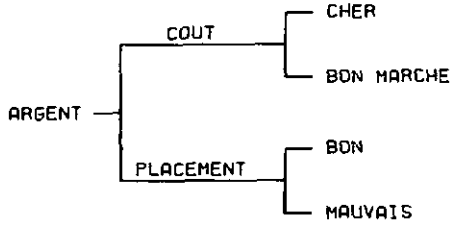


Figure 3.5

finalement, on tentera de détailler un peu plus le troisième système, celui de la "santé". Il s'agit d'un système simultané comportant trois sous-syatèmes. Le premier sous-système désigné par "concerne", indique si la raison se rapporte à "moi", aux "autres" ou à tout le monde en général, auquel cas on utilisera le terme "indéterminé". Le deuxième sous-système, désigné par "avis", indique si la fumée est considérée comme "positive" (bon, agréable), "négative" (mauvais, désagréable, dangereux) ou "neutre" pour la santé. Enfin, le troisième sous-système se rapporte directement à la santé. Lorsqu'on parle de santé, il conviendrait de faire la différence entre santé physique et santé psychique. Cependant, la frontière entre le physique et le psychique n'est pas facile à distinguer, du moins pour quelqu'un qui n'est pas psychologue. En effet, lorsque quelqu'un dit : "je ne supporte pas la fumée", il n'est pas facile de savoir s'il s'agit d'un désagrément physique ou psychique. On ne considérera donc, assez grossièrement, que trois catégories. La santé "en général" désignera la santé tant du point de vue physique que psychique. Lorsqu'on pourra clairement trancher, si l'on parle d'un "organe" ou d'un "sens", on entrera dans le système santé "physique". On renoncera, par contre, à détailler les systèmes "organe" (*système respiratoire, système vasculaire,* syatème nerveux, etc...) et "sens" (goût, odorat, etc...). Enfin, dans tous lea autres caa, on parlera de la santé "en particulier". On conviendra de garder, dans ce dernier système, les termes figurant dans le corpus, en vue d'une éventuelle analyse ultérieure plus détaillée. Le système "en particulier" comprendra donc des termea comme aimer, ne pas aimer, ne pas aupporter, etc... Le aystème "santé", tel qu'on vient de le définir, est représenté dena la figure 3.6.

Le corpua.

Le corpus ae constitue des réponses que les gens ont donné à la question "Fumez-vous? ". Toutes lea réponses ont été gardées et sont présentées ici dans un ordre arbitraire. De plus, elles ont été gardéea rigoureusement telles qu'elles ont été fournies, à l'exception de certaines fautes orthographiques.

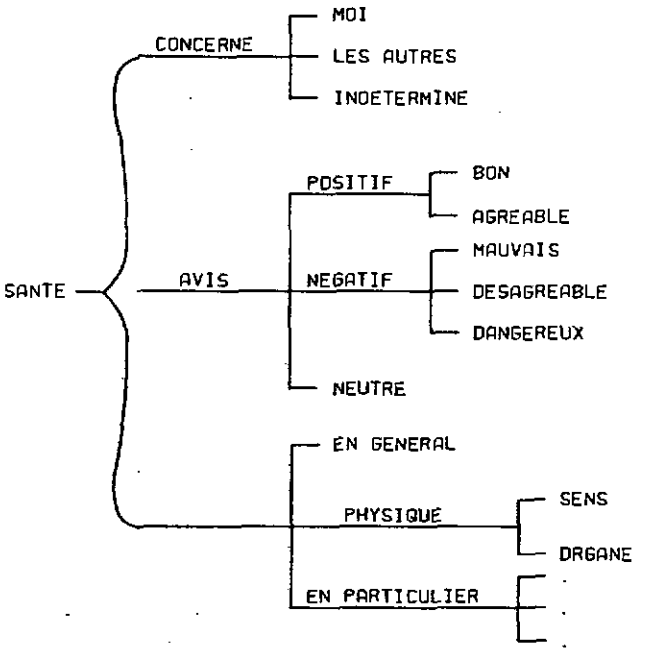


Figure 3.6

A) Je ne fume pas.
Pour des raisons de santé essentiellement : les risques sont trop grands et je supporte mal la fumée.
Pour des raisons financières ensuite : je trouve qu'il vaut mieux "investir" ailleurs.
Enfin, la fumée me gêne et j'estime qu'il ne faut pas l'imposer aux autres.

B) Non, jamais.
Utilité, bien-être pas évidents.

Pas de fumeur dans la famille. Cela aide.
A 15-16 ans, je prenais le contre-pied de mes copains qui eux, voulaient s'essayer à la fumée.

C) Non.
Après avoir fumé pendant quelques mois, j'ai vu que cela ne m'apportait rien. De plus, j'ai vu que j'avais quelques problèmes de souffle dans les différents sports que je faisais. Tout ceci m'a aidé à cesser immédiatement et définitivement.

D) Oui, un paquet par jour.
J'ai commencé, je continue, mais je n'aime pas. J'attend l'occasion d'arrêter (à nouveau).

E) Non.
Je n'aime pas.

F) Je ne fume pas.
Je pratique un sport en compétition. Si le fait de fumer est une mode pour certains, il est aussi un moyen de se calmer. Or, je n'en ressens absolument pas le besoin, étant calme de nature.

G) Je fume depuis un certain nombre d'années. J'ai commencé par plaisir : à rien n'est comparable le frottement de la nicotine dans les muqueuses buccales et il faut l'avoir éprouvé avant de se prononcer pour ou contre le tabac. Je ne fume pas que par plaisir, mais aussi vraisemblablement pour des raisons sociales. Toutefois, j'ai essayé d'arrêter de fumer à plusieurs reprises, mais à plusieurs reprises j'ai recommencé de plus belle et je me suis retrouvé à fumer un paquet par jour. Ainsi donc, je vais essayer de trouver un moyen efficace pour arrêter enfin : je cherche des succédanés.

H) Non.
J'en ai jamais senti le besoin et la fumée me brûle les yeux.
Je ne supporte pas la fumée ! Ouvrez les fenêtres !

I) Certes, je fume et beaucoup, c'est-à-dire dès le matin et jusqu'au soir. La pipe exclusivement ou quasi.
Pourquoi ? Par habitude évidemment. Mais l'habitude doit naître. Comment cela a-t-il commencé ? Imitation sans doute. Et le plaisir.

J) Non.
Raisons de santé et de salubrité (dégâts causés par la fumée aux parois, rideaux, oeuvres d'art et autres).

K) Oui, très souvent.
C'est une façon d'ennuyer un cinquante pour cent de la population : ceux qui ne fument pas.

L) Je ne fume plus depuis deux ans.
J'ai arrêté pour des raisons médicales, mais malheureusement, dans le bureau où je travaille j'ai l'impression de fumer plus d'un paquet par jour en raison de la proximité de mon collaborateur.

M) Oui, je fume, moyennement à beaucoup, selon les circonstances.
La raison ? je pense qu'il est inutile de déblatérer longtemps sur les raisons initiales qui m'ont incité un jour à entreprendre cette activité. Aujourd'hui, j'ai le sentiment que chaque cigarette permet de marquer un micro-événement de ma vie quotidienne : ouvrir un bouquin, un journal, prendre la parole, marquer la fin d'un repas, prendre une décision, etc...

N) Oui, parfois.
C'est une occupation intéressante et çà développe le goût.

La codification.

Considérons le réseau complet de la figure 3.7 qui appelle certains commentaires. On constate d'emblée le rôle important joué par la notation. En d'autres termes plusieurs pages seraient nécessaires pour exprimer "en prose" tout ce que dit le

réseau, alors que, grâce au formalisme utilisé, il ne suffit que d'une seule page. Par ailleurs, il convient de souligner le côté explicite qu'apporte ce formalisme. Ce dernier aspect permettant de critiquer et de discuter l'analyse de manière précise et non ambiguë. Là encore, si l'on avait affaire à de la prose, il serait extrêmement difficile de détecter les points forts et les imperfections.

Chaque réponse sera codée à l'aide des termes du réseau complet. Afin de ne pas trop allonger le code, on conviendra, dans la plupart des cas, de ne pas faire précéder les termes par leur condition d'entrée.

```
A) FUMEE(NON(jamais)
     RAISONS(SANTE(indéterminé dangereux en général))
     RAISONS(SANTE(moi désagréable
                   en particulier(pas supporter)))
     RAISONS(ARGENT(PLACEMENT(mauvais)))
     RAISONS(SANTE(moi désagréable en particulier(gêner)))
     RAISONS(SANTE(les autres désagréable
                   en particulier(gêner))))
```

"Je ne fume pas" est donc codé NON(jamais), sous-entendu "actuellement je ne fume pas" et "je n'ai jamais fumé", aucune indication ne permettant de supposer que A a été fumeur.

"Pour des raisons de santé essentiellement:" annonce quelles seront les raisons de santé, on n'en tiendra donc pas compte. Ces raisons sont: "les risques sont trop grands" qui sera codé SANTE(indéterminé dangereux en général); "indéterminé" puisque A ne dit pas que les risques sont grands pour lui en particulier ou pour les autres; "dangereux" traduit le concept de risque; "en général" puisque A ne parle pas précisément de la santé physique et qu'il n'utilise pas de terme particulier.

La deuxième raison code la phrase "je supporte mal la fumée".

ARGENT(PLACEMENT(mauvais)) code la phrase "Pour des raisons financières ensuite: je trouve qu'il vaut mieux "investir" ailleurs".

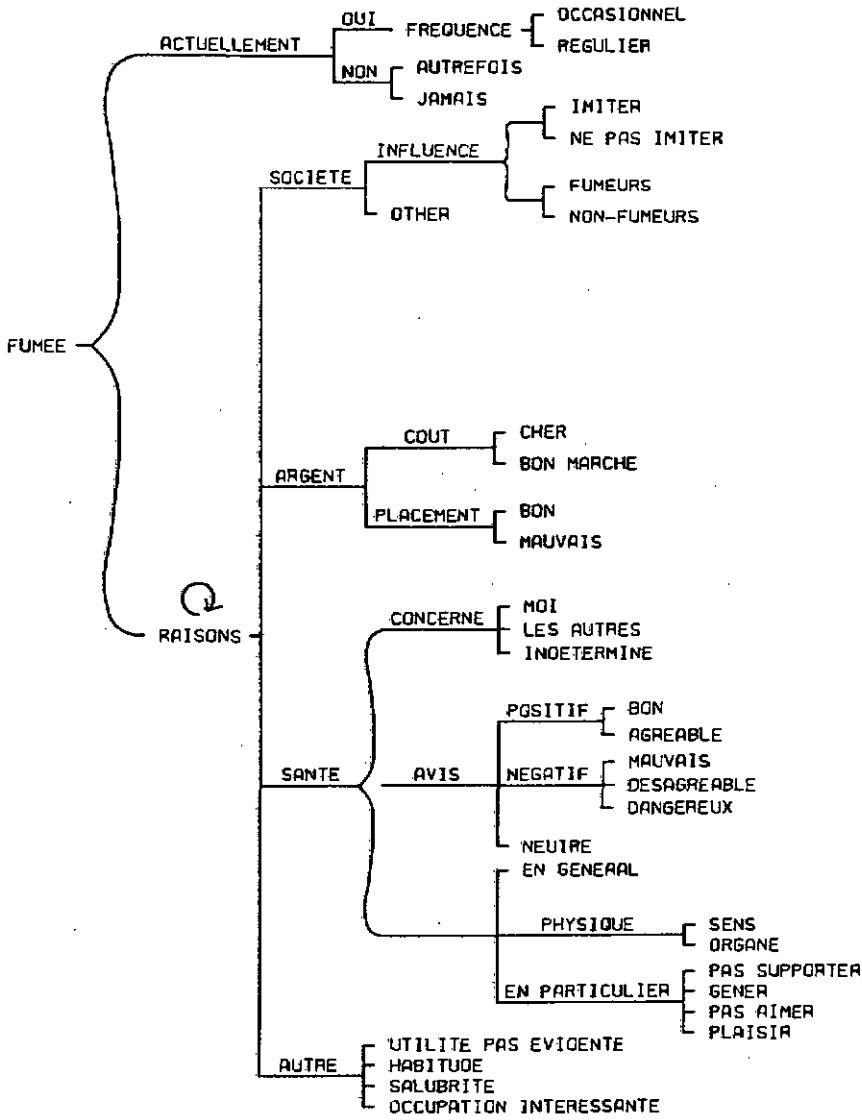

FUMEE
ACTUELLEMENT
OUI
FREQUENCE
OCCASIONNEL
REGULIER
NON
AUTREFOIS
JAMAIS
SOCIETE
INFLUENCE
IMITER
NE PAS IMITER
OTHER
FUMEURS
NON-FUMEURS
RAISONS
ARGENT
COUT
CHER
BON MARCHE
PLACEMENT
BON
MAUVAIS
SANTE
CONCERNE
MOI
LES AUTRES
INDETERMINE
AVIS
POSITIF
BON
AGREABLE
NEGATIF
MAUVAIS
DESAGREABLE
DANGEREUX
NEUTRE
EN GENERAL
PHYSIQUE
SENS
ORGANE
EN PARTICULIER
PAS SUPPORTER
GENER
PAS AIMER
PLAISIR
AUTRE
UTILITE PAS EVIDENTE
HABITUDE
SALUBRITE
OCCUPATION INTERESSANTE

Figure 3.7

La qustrième raison code la phrase "la fumée me gêne".

Enfin, le phrase "j'estime qu'il ne faut pss imposer la fumée aux autres" n'a pas été prévue dans le réseau, puisqu'il ne s'sgit pss d'une raison qui pousse A à ne pas fumer. Normalement, on devrait laisser tomber purement et simplement cette information. On décidera cependant d'en tenir compte en la traduisant psr l'idée "la fumée gêne les autres" c'est-à-dire SANTE(les autres désagréable en psrticulier(gêner)).

```
B) FUMEE(NON(jamais)
     RAISONS(AUTRE(utilité pas évidente))
     RAISONS(SANTE(indéterminé neutre
                   en général (*bien-être pas évident*)))
     RAISONS(SOCIETE(imiter non-fumeurs))
     RAISONS(SOCIETE(ne pas imiter fumeurs)))

C) FUMEE(NON(autrefois)
     RAISONS(SANTE(moi neutre en général (*n'apporte rien*)))
     RAISONS(SANTE(moi mauvais
                   physique(organe) (*problèmes de souffle*))))

D) FUMEE(OUI(régulier)
     RAISONS(SANTE(moi désagréable en particulier(pas aimer))))

E) FUMEE(NON(jamais)
     RAISONS(SANTE(moi désagréable en particulier(pas aimer))))

F) FUMEE(NON(jamais)
     RAISONS(SOCIETE(other (*mode pour certains*)))
     RAISONS(SANTE(indéterminé bon
                   physique(organe) (*les nerfs*))))
     RAISONS(SANTE(moi neutre
                   en général (*n'en ressent pas le besoin*))))

G) FUMEE(OUI(régulier)
     RAISONS(SANTE(moi agréable en psrticulier(pisisir)))
     RAISONS(SANTE(indéterminé agréable
                   physique(sens) (*goût*)))
     RAISONS(SOCIETE(other)))
```

```
M) FUMEE(NON(jamais)
     RAISONS(SANTE(moi neutre
                   en général (*n'en ressent pas le besoin*)))
     RAISONS(SANTE(moi mauvais physique(organe) (*yeux*)))
     RAISONS(SANTE(moi désagréable
                   en particulier(pas supporter))))

I) FUMEE(OUI(régulier)
     RAISONS(AUTRE(habitude))
     RAISONS(SOCIETE(imiter fumeurs))
     RAISONS(SANTE(indéterminé agréable
                   en particulier(plaisir))))

J) FUMEE(NON(jamais)
     RAISONS(SANTE(indéterminé mauvais en général))
     RAISONS(AUTRE(salubrité (*dégâts aux objets*))))

K) FUMEE(OUI(régulier)
     RAISONS(SANTE(les autres désagréable en général)))

L) FUMEE(NON(autrefois)
     RAISONS(SANTE(moi mauvais en général)))

M) FUMEE(OUI(régulier)
     RAISONS(AUTRE(habitude)))

N) FUMEE(OUI(occasionnel)
     RAISONS(AUTRE(occupation intéressante))
     RAISONS(SANTE(indéterminé bon physique(sens) (*goût*))))
```

La vérification.

La démarche qui nous a conduits jusqu'à ce stade de l'analyse se révèle être purement intellectuelle, c'est-à-dire ne fait appel essentiellement qu'à la réflexion. Par contre, la phase de vérification pourra être effectuée automatiquement par ordinateur et c'est précisément, entre autres, ce dont se chargera le programme BARBARA.

La vérification consiste principalement en deux choses. D'une part, à vérifier la cohérence du réseau et d'autre part à s'assurer que chaque élément du code n'est toujours exprimé qu'à l'aide de termes appartenant au réseau et est bien une des réalisations possibles de ce réseau. Il convient donc, pour réaliser cette double vérification, d'introduire dans le programme BARBARA, d'une part le réseau et d'autre part le code qui est censé lui correspondre.

Sans entrer dans les détails de fonctionnement du programme, qui seront précisés aux chapitres suivants, disons simplement qu'il s'agit d'un programme interactif, c'est-à-dire basé sur un dialogue utilisateur - machine, permettant, dans sa première phase, d'introduire un réseau systémique quelconque à l'aide des cinq constructeurs : BAR, BRA, REC, CON et ANY.

Le réseau de la figure 3.7 serait introduit dans le programme à l'aide de l'ensemble des instructions suivantes:

```
fumee = BRA(actuellement REC(raisons));
actuellement = BAR(oui non);
oui = frequence;
frequence = BAR(occasionnel regulier);
non = BAR(autrefois jamais);
raisons = BAR(societe argent sante autre);
influence = BRA(BAR(imiter ne_pas_imiter)
                BAR(fumeurs non_fumeurs));
argent = BAR(cout placement);
cout = BAR(cher bon_marche);
placement = BAR(bon mauvais);
sante = BRA(concerne avis
            BAR(en-general physique en-particulier));
concerne = BAR(moi les_autres indetermine);
avis = BAR(positif negatif neutre);
positif = BAR(bon agreable);
negatif = BAR(mauvais desagreable dangereux);
physique = BAR(sens organe);
societe = BAR(influence other);
autre = BAR(utilite habitude salubrite occupation);
en_particulier = BAR(pas_supporter gener pas_aimer plaisir);
```

A noter que:

- tout élément du réseau est désigné à l'aide d'un identificateur. Un identificateur se constitue toujours d'une chaîne quelconque de lettres (non comprises les lettres munies d'accent), de chiffres ou des trois symboles spéciaux - ("trait d'union"), _ ("souligné") et / ("barre" soit slash). Le premier caractère d'un identificateur est toujours une lettre. Deux identificateurs pouvant être séparés par un ou plusieurs blancs, un identificateur ne comprendra donc jamais de blancs. Ceci explique certaines différences entre les noms qui figurent dans le réseau et ceux introduits dans le programme.

- l'utilisation de minuscules et de majuscules n'est pas significative. On n'a utilisé des majuscules que pour mettre en évidence les constructeurs BAR, BRA et REC.

- le programme aide l'utilisateur à construire son réseau en lui permettant de savoir en tout temps quels sont les éléments qui ont déjà été introduits, où et combien de fois ils apparaissaient, dans le cas d'une condition d'entrée, comment elle est définie, etc...

En ce qui concerne la vérification de la cohérence du réseau, le programme se chargera d'empêcher des définitions différentes pour une même condition d'entrée, signale l'utilisation d'éléments qui provoqueraient la récursivité, vérifie la connexité du réseau, etc...

Dans sa deuxième phase, le programme permet d'introduire le code qui est censé correspondre au réseau. A un petit détail syntaxique près qui veut que l'on donne A = FUNEE(... plutôt que A) FUNEE(..., le code sera introduit dans le programme exactement tel qu'il a été donné sous le paragraphe "codification".

Le programme vérifie que chaque élément qu'on lui fournit est bien un élément appartenant au réseau, que la suite de ces éléments constitue bien une réalisation possible du réseau, etc...

Là aussi, le programme aide l'utilisateur à introduire le code en lui permettant en tout temps de savoir quelles sont les réponses qui ont déjà été fournies, comment elles l'ont été, comment elles doivent l'être, etc...

L'analyse réseau-code.

S'il est éventuellement possible d'envisager la vérification du réseau et du code, de manière non automatique, grâce à un travail minutieux et fastidieux, il n'en va pas de même en ce qui concerne l'analyse du code par rapport au réseau. Le programme BARBARA va donc jouer là un rôle prépondérant, puisqu'il permettra d'analyser et d'interroger l'ensemble réseau-code, essentiellement à l'aide de trois fonctions: LIST, NUM et TAB.

Dans l'exemple que nous avons traité, le réseau est évidemment beaucoup trop simplifié et l'ensemble des réponses (quatorze) beaucoup trop limité pour pouvoir procéder à une analyse d'un très haut intérêt. Cependant, nous allons donner quelques exemples du genre des questions que l'on pourrait poser.

- Avant toute chose, il paraît intéressant de savoir combien de réponses ont été codées et quelles sont les personnes qui fument et quelles sont celles qui ne fument pas.

  A la question NUM(fumee) le programme répondra 14, puisque 14 réponses ont été codées et que toutes satisfont la condition d'entrée principale du réseau: "fumée". A NUM(oui) le programme répondra 6 et à NUM(non), 8. En effet, 6 personnes prétendent fumer et 8 affirment ne pas fumer.

- La fonction LIST donne la liste des réponses qui satisfont un certain critère. On pourrait, par exemple, s'intéresser à l'ensemble des fumeurs occasionnels et à ceux qui ont arrêté de fumer, afin de retrouver dans le corpus les termes précis qu'ils ont utilisés. On posera alors la question LIST(occasionnel or autrefois) et le programme répondra: C, L et N.

- L'utilisation de tableaux croisés fournit souvent des renseignements intéressants. Par exemple, à la question TAB(negatif(mauvais) desagreable dangereux , oui non) le programme fournira le tableau suivant:

	negatif(mauvais)	desagreable	dangereux
oui	0	2	0
non	4	3	1

On remarquera que le terme "mauvais" apparaît deux fois dans le réseau (sous la condition d'entrée "placement" et sous la condition d'entrée "négatif"). C'est pourquoi il est nécessaire de le qualifier par sa condition d'entrée, afin de le désigner de manière non ambiguë.

Dans le tableau ci-dessus, il est intéressant de constater que les fumeurs ont l'air de faire peu de cas de leur santé alors que les non-fumeurs semblent y attacher une plus grande importance.

- Seul deux fumeurs invoquent les aspects négatifs de la fumée sur la santé. Encore pourrait-on se poser la question de savoir s'il s'agit de leur propre santé: LISI(oui and desagreable and (les-autres or indetermine)) donne K. En effet, K reconnaît que la fumée est désagréable essentiellement pour les autres, puisqu'il fume pour ennuyer les non-fumeurs.

En tentant de mettre en évidence certaines caractéristiques, à l'aide des opérateurs and, or et not, on pourrait, bien sûr, multiplier les exemples et pousser l'analyse suffisamment loin pour en tirer des conclusions générales. On se gardera cependant de le faire, étant donné l'imprécision du réseau et la pauvreté de l'échantillon.

Pour terminer, signalons les annexes II et III comme compléments à ce chapitre. On y trouve la description (en anglais) de deux recherches effectuées au Chelsea College de l'Université de Londres. On aurait pu citer de nombreux travaux qui utilisent ou ont utilisé les réseaux systémiques. Nous nous bornerons cependant à n'en mentionner que deux qui l'un et l'autre s'appuient

sur BARBARA. Par ailleurs, l'emploi qu'ils font des réseaux systémiques est de nature fort diverse; ils illustrent ainsi la variété des domaines dans lesquels la méthode décrite est applicable.

L'annexe II décrit une recherche qui se situe dans un contexte pouvant s'apparenter aux banques de données. Le matériel utilisé dans le cadre de cette expérience consiste en quelques centaines de questions issues d'examens de physique. Le réseau représente les différents domaines de la physique relatifs aux questions posées. Il est clair que ce réseau ne saurait représenter toute la science physique. Mais il offre l'avantage de pouvoir être affiné en fonction de l'évolution des examens, sans pour autant perdre des informations antérieures.

L'enregistrement des différentes questions en fonction du réseau constitue une "banque de données" dont on peut extraire des informations sélectives et condensées. A long terme, il devrait être possible, à partir de cette banque, de produire de nouvelles questions d'examen et même de pondérer leur degré de difficulté en tenant compte des résultats obtenus lors d'examens précédents.

La seconde recherche relève essentiellement de la psychologie; elle vise à étudier l'image que des écoliers se font d'eux-mêmes face à l'apprentissage scolaire. Cette recherche comporte deux parties qui chacune donne lieu à un réseau. Le premier réseau (expliqué dans l'annexe III) a pour base un questionnaire ("avec qui joues-tu?", "quel est ton meilleur ami?", "que penses-tu de lui?", etc...) dont le but est d'établir toute une série de catégories et de relations entre les élèves d'une même classe (garçons-filles, leaders, groupes d'amitié, etc...). Le second réseau (en cours d'élaboration) "schématise" un ensemble d'interviews au cours desquelles on a posé des questions pour savoir si un élève était bon, moyen ou mauvais (selon l'élève lui-même, selon ses professeurs et selon les autres enfants). On s'est, par ailleurs, intéressé aux critères qu'utilise l'interviewé pour considérer un élève comme plutôt bon, respectivement plutôt mauvais.

Etant donné la complexité du matériel, cette recherche n'aurait certainement pas été réalisable sans l'aide des réseaux systémiques. De plus, un traitement par ordinateur s'est avéré indispensable, car chaque classe étudiée donne lieu à plusieurs milliers d'informations élémentaires.

REFERENCES

1. WEIZENBAUM, J., Computer Power and Human Reason: From Judgment to Calculation. San Francisco, Freeman, 1976.

2. TURNER, G.J., Social class and children's language of control at age 5 and age 7. in Bernstein, B. (Ed.), Class, Codes and Control, Vol 2. London, Routledge and Kegan Paul, 1972.

3. OGBORN, J. (Ed.), Small Group Teaching in Undergraduate Science. Heinemann Educational Books, London, 1977.

4. BLISS, J. et OGBORN, J., Students' Reaction to Undergraduate Science. Heinemann Educational Books, London, 1977.

5. BLISS, J. et OGBORN, J., The analysis of Qualitative Data. Eur. J. Sc. Educ., 1979, 1, 427-440.

CHAPITRE QUATRIEME

LE PROGRAMME BARBARA

D'une manière générale, dans ce chapitre, les identificateurs (généralement en anglais) qui figurent entre parenthèses dans le texte, se réfèrent aux principales procédures du programme BARBARA qui réalisent les fonctions décrites.

Le programme BARBARA est un programme entièrement interactif qui se déroule en trois phases:

1) Introduction d'un réseau systémique.
   Lors de cette première phase, le programme construit en mémoire un graphe qui correspond au réseau systémique, tel que l'introduit l'utilisateur. Voir paragraphe 4.3.

2) Introduction et vérification du code.
   Cette deuxième phase a pour but de vérifier que le code correspond bien au réseau systémique tel qu'il a été défini lors de la première phase. En outre, au fur et à mesure de l'introduction du code, le programme mémorise sur le graphe tous les chemins empruntés par ce code. Voir paragraphe 4.4.

3) Interrogation de l'ensemble réseau-code.
   Cette dernière phase permet l'exploitation des données telles qu'elles ont été construites au cours des deux premières phases. Voir paragraphe 4.5.

4.1 Le programme BARBARA comme compilateur de compilateurs

On a vu, en 2.2, qu'un réseau systémique exprimé à l'aide des seuls systèmes simples, -bar et -bra pouvait être considéré comme une grammaire indépendante de contexte. Dès lors, si l'on considère tout réseau systémique comme une grammaire indépendante de contexte (le cas des systèmes-con et -any sera examiné à part), la deuxième phase du programme BARBARA consiste, entre autres, à vérifier que le code est bien une dérivation possible dans cette grammaire. Il s'agit là, typiquement, de l'une des tâches réalisées par un compilateur de langage, d'un langage de programmation par exemple.

En effet, un compilateur de langage de programmation est un programme qui réalise principalement deux fonctions:

1) il vérifie qu'un programme est en tous points conforme à la syntaxe et à la sémantique du langage qu'il est censé compiler.

2) il produit du code, c'est-à-dire un ensemble d'instructions capables d'être comprises par l'ordinateur pour lequel il a été écrit. Ce deuxième aspect ne nous intéresse pas du tout dans le cas du programme BARBARA.

Un compilateur de langage de programmation est, en général, écrit pour un langage bien défini. Dans le cas de BARBARA, on rencontre une difficulté supplémentaire, qui est que le langage à "compiler" est, a priori, quelconque, pourvu qu'il soit indépendant de contexte. Un compilateur qui n'est pas écrit pour un langage bien défini, mais auquel on commence par donner la grammaire du langage qu'il doit compiler, est connu sous le terme de compilateur de compilateurs. A l'exception de la production du code pour un ordinateur donné, le programme BARBARA peut donc être considéré comme un compilateur de compilateurs.

Il existe, évidemment, une théorie extrêmement vaste sur la manière de compiler (1), qui se concrétise principalement autour de deux techniques:

1) la technique dite LR(k) (analyse de gauche (left) à droite (right) avec k symboles d'avance) ou technique "bottom-up" qui consiste à remonter jusqu'à l'axiome de la grammaire à partir de la chaîne à analyser (2).

2) la technique LL(k) ou "top-down" ou encore par "descente récursive" qui consiste à atteindre, à partir de l'axiome, les éléments de la chaîne à analyser (3).

La technique utilisée dans le cas du programme BARBARA est la technique LL, plus particulièrement LL(1), c'est-à-dire avec un

symbole d'avance. La technique LL(1) est certainement la plus utilisée actuellement. En particulier, depuis que N. WIRTH (4) l'a clairement précisée en développant le compilateur PASCAL, langage dans lequel le programme BARBARA est écrit (5).

4.2 Analyseurs LL(1)

Pour pouvoir utiliser une technique de compilation LL(1), c'est-à-dire programmer un analyseur LL(1), il est nécessaire que la grammaire que l'on désire compiler, satisfesse un certain nombre de conditions, qui en font une grammaire dite LL(1).

A partir des notations introduites en 2.1, on va définir les deux fonctions FIRST et FOLLOW sur une grammaire G = (N,T,P,S).

On rappelle que:

N est le vocabulaire non-terminal.
T est le vocabulaire terminal.
P désigne l'ensemble des productions.
S est l'axiome de la grammaire.
V^* dénote l'ensemble de toutes les chaînes formées à l'aide des symboles de V = N ∪ T.

La fonction FIRST.

Considérons une chaîne quelconque x de V^*. Alors la fonction FIRST est définie comme suit:

- si $x = ay$ avec $a \in T$ et $y \in V^*$ alors $FIRST(x) = \{a\}$

- si $x = Ay$ avec $A \in N$, $y \in V^*$ et $A \rightarrow z_1|z_2|\ldots|z_n$
  alors $FIRST(x) = FIRST(z_1) \cup FIRST(z_2) \cup \ldots \cup FIRST(z_n)$.

La fonction FIRST définit donc l'ensemble de tous les symboles terminaux qui peuvent apparaître en première position d'une chaîne donnée.

La fonction FOLLOW.

Soit $A \in N$. Considérons l'ensemble de toutes les productions de P où A apparaît à droite du symbole de dérivation $\rightarrow$:

$$A_1 \rightarrow x_1 A y_1$$
$$A_2 \rightarrow x_2 A y_2$$
$$\vdots$$
$$A_n \rightarrow x_n A y_n$$

avec $A_i \in N$ et $x_i, y_i \in V^*$ pour $i=1,2,\ldots,n$.

Alors $FOLLOW(A) = FIRST(y_1) \cup FIRST(y_2) \cup \ldots \cup FIRST(y_n)$.

Définition: Une grammaire G = (N,T,P,S) est une grammaire LL(1) si et seulement si elle satisfait les deux conditions suivantes:

condition 1: toute production, appartenant à P, de la forme:
$A \rightarrow x_1 \mid x_2 \mid \ldots \mid x_n$
doit être telle que:
pour tout x_i et $x_j \neq \epsilon$ (chaîne vide) on ait:
$FIRST(x_i) \cap FIRST(x_j) = \emptyset$ pour tout $i \neq j$.

condition 2: tout symbole non-terminal $A \in N$ qui est susceptible d'engendrer la chaîne vide ϵ, doit être tel que:
$FIRST(A) \cap FOLLOW(A) = \emptyset$.

Intuitivement, on comprend fort bien ce que ces deux conditions impliquent du point de vue de l'analyse. Si le non-terminal A se dérive en plusieurs productions différentes, alors on doit être capable de décider immédiatement laquelle de ces productions choisir, en fonction du symbole de la chaîne que l'on est en train d'analyser.

Exemples:

1) Considérons la grammaire donnée par les quatre productions:

$$S \rightarrow a \mid A$$
$$A \rightarrow ay \mid b$$

On a:
FIRST(ax) = {a}
FIRST(A) = FIRST(ay) ∪ FIRST(b) = {a,b}

Comme FIRST(ax) ∩ FIRST(A) = {a} ≠ ∅, cette grammaire ne satisfait pas la condition 1. Il est trivial de trouver une grammaire équivalente qui satisfasse toujours la condition 1. Il suffit d'introduire une nouvelle règle qui réunit les éléments communs. Dans notre cas, ce serait:

$$S \rightarrow aC \mid A$$
$$C \rightarrow x \mid y$$
$$A \rightarrow b$$

2) Supposons qué l'on désire engendrer le langage:
L = {a,aa,aaa,...}.
Une grammaire capable d'engendrer L pourrait être donnée par: S → a|Sa. On notera que cette grammaire est récursive à gauche. En outre, elle ne satisfait pas la condition 1. Cependant, il est facile de la transformer en: S → Ca et C → ε|S pour qu'elle satisfasse la première condition. Malheureusement, cette dernière grammaire ne satisfait pas la condition 2. En effet, à partir de S, on ne peut dériver que des chaînes de la forme a,aa,aaa,..., donc FIRST(S) = {a}. Par ailleurs, on a que FIRST(C) = FIRST(S) et comme FOLLOW(C) = {a}, on en tire que FIRST(C) ∩ FOLLOW(C) = {a} ≠ ∅.
La seconde condition interdit donc la récursivité à gauche. Par contre, on se convaincra aisément que la récursivité à droite peut être LL(1). En effet, dans le cas du langage L, il suffit de considérer la grammaire S → a|aS, qui se transforme en: S → aC et C → ε|S, dont on vérifie facilement qu'elle est LL(1).

En conclusion, on pourrait postuler que, pour exprimer la récursivité, il suffit de n'utiliser que la récursivité à droite. Cependant, on préfère introduire une notation supplémentaire {u} qui désigne l'ensemble des séquences: u,uu,uuu,.... Ceci d'autant plus que du point de vue du programme BARBARA, il n'existe pas de notion équivalente à la chaîne vide, ϵ.

Dans BARBARA, la notation {u} se traduit par REC(u). A noter que pour certains auteurs, {u} désigne la séquence: ϵ,u,uu,uuu... et non la séquence u,uu,uuu,.... On soulève parfois l'ambiguïté en désignant la première séquence par $\{u\}^*$ et la seconde par $\{u\}^+$. Dans le cas de BARBARA, il n'y a jamais d'ambiguïté, car REC(u) désigne toujours u,uu,uuu....

4.3 Traitement du réseau (READNETWORK)

Le programme BARBARA, on l'a dit, a été entièrement écrit en PASCAL.
Il est bien évident que la taille d'un réseau ou du code, peut varier considérablement d'une application à l'autre. C'est la raison pour laquelle il est indispensable de pouvoir adapter la taille du programme en fonction de la taille du réseau, respectivement du code. PASCAL offre la possibilité d'allocation dynamique de mémoire (demande de mémoire en cours d'exécution du programme) grâce à la notion de pointeurs. Toutes les données, dans le programme BARBARA, sont donc maintenues dans des structures d'informations dynamiques, telles que des listes, des arbres, etc.... Cette approche, outre qu'elle permet d'adapter la taille du programme au volume des données, offre un avantage supplémentaire: elle n'impose aucune limite, du moins, aucune limite pratiquement accessible, quant à la taille et à la complexité du réseau, respectivement du code.

Cependant, il est évident que BARBARA est appelé à s'exécuter sur un ordinateur donné et qu'il est donc limité par la taille de la mémoire disponible sur l'ordinateur (à moins qu'il ne s'agisse d'un ordinateur à mémoire virtuelle, auquel cas on dispose d'une mémoire pratiquement "illimitée").

A l'origine, BARBARA a été développé sur un ordinateur COC de la série CYBER. Cette machine se caractérise par le fait qu'un mot de mémoire centrale s'étend sur soixante bits. Afin d'utiliser au mieux la place disponible en mémoire, il a toujours été tenu compte de cette caractéristique. En particulier, les champs des enregistrements paquetés (packed record) sont définis (à l'aide de constantes) de manière à toujours remplir entièrement un mot machine. Cela explique, également, que deux identificateurs sont distincts, si leurs dix premiers caractères le sont. L'ensemble des constantes qui définissent les champs des enregistrements, la longueur d'un identificateur, etc... pourront facilement être modifiés en fonction de l'ordinateur sur lequel BARBARA sera appelé à être exécuté.

Notons, au passage, que le programme est actuellement installé à Londres sur une COC (système d'exploitation NOS) mais maintenu et étendu sur une VAX 11/780 (système d'exploitation VMS). Ceci oblige à toujours écrire les choses de telle manière que l'on puisse garantir un maximum d'indépendance par rapport à l'ordinateur.

Les symboles.

Les symboles sont l'ensemble des identificateurs qui apparaissent dans le réseau. Il peut s'agir de conditions d'entrée (non-terminaux) ou de termes (terminaux). Les symboles sont maintenus dans un arbre de recherche. Un arbre de recherche est un arbre binaire dont les éléments peuvent être retrouvés à l'aide d'une clé unique. Chaque noeud n_i de l'arbre est tel que toutes les clés du sous-arbre gauche de n_i sont inférieures ou égales à la clé de n_i et que toutes celles du sous-arbre droit sont supérieures à la clé de n_i. Dans le cas de BARBARA, la clé est l'identificateur lui-même, c'est-à-dire, une chaîne de caractères.

Tout symbole peut être soit un non-terminal soit un terminal. Une donnée définissant un "morceau" de réseau, comme S = bar(bra(a b) c), est appelée une règle de définition du réseau. Lorsqu'on l'introduit dans le programme (READRULE), une règle se termine toujours par un point-virgule (;). Un symbole apparais-

sant à gauche du signe = d'une règle est toujours considéré comme non-terminal. Deux cas doivent être envisagés, à l'occurrence d'un symbole:

1) Le symbole ne figure pas encore dans l'arbre de recherche.
   Il y est alors introduit en bonne place (SEARCHSYMB), c'est-à-dire en respectant la relation d'ordre définie sur l'arbre, et, s'il s'agit d'un non-terminal, en préparant un pointeur vers le "morceau" de réseau qu'il va définir. A noter qu'un symbole qui apparaît à la droite du signe = d'une règle et qui ne figure pas déjà dans l'arbre, sera toujours, a priori, considéré comme terminal.

2) Le symbole figure déjà dans l'arbre de recherche.
   Deux sous-cas sont à envisager:

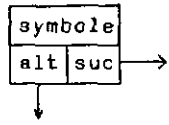
   - Le symbole figure dans l'arbre comme terminal.
     Si le symbole rencontré est à droite du signe =, on augmente un compteur associé à chaque symbole qui indique combien de fois il apparaît dans le réseau. Si par contre, il apparaît à gauche du signe =, c'est qu'il s'agit d'un non-terminal, on le fait donc passer à l'état de non-terminal (préparation d'un pointeur vers le "morceau" de réseau qu'il définit) ce qui a comme conséquence de "connecter" à la nouvelle règle l'ensemble de toutes les règles dans lesquelles le symbole apparaissait déjà.

   - Le symbole figure dans l'arbre comme non-terminal.
     Si le symbole rencontré apparaît à droite du signe =, c'est qu'il s'agit préciséement d'une référence à ce non-terminal (augmentation du compteur des références). Si par contre, il apparaît à gauche du signe =, c'est qu'il s'agit soit d'une erreur, soit d'une volonté délibérée de l'utilisateur de redéfinir différemment un non-terminal. Le programme se fait confirmer ce dont il s'agit. Il convient d'être conscient des implications que la redéfinition d'une règle peut avoir sur le réseau: "déconnexion" d'un certain nombre de règles, destruction de systèmes-con ou -any, etc.... Du point de vue du programme, il s'agit d'une opération relativement complexe nécessitant la destruction de l'an-

cien "morceau" de réseau (RETNET), la destruction éventuelle de certains syatèmes-con ou -any (DELETECOND) - tout système détruit aera évidemment signalé (WRITECOND) -, la mise à jour de l'arbre contenant les aymboles et dans certains cas, la suppression de symboles (ceux pour lesquels le compteur dea références tombe à zéro), tout en maintenant la relation d'ordre aur l'arbre (DELETESYMB), etc...

Le réseau.

La technique utilisée pour représenter le réseau s'inspire très largement d'une méthode aervant à représenter des grammaires indépendántes de contexte, en vue de développer un compilateur de compilateurs (6). Chaque symbole est introduit dans un noeud et les noeuds sont connectés entre eux par deux pointeurs: l'un, le succeaseur, désigne le aymbole qui doit suivre, l'autre, l'alternative, désigne la liste des symboles qui peuvent être choisis "optionnellement". Schématiquement, un noeud serait repréaenté comme suit:



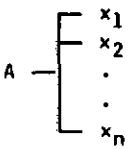
Dana notre cas, on notera que ce n'est pas le symbole lui-même qui est contenu dans le noeud, mais un pointeur vers le symbole, ce dernier étant maintenu dans l'arbre de recherche. Cependant, afin de ne pas alourdir les schémas qui suivent, on inscrira dans le noeud le aymbole lui-même et non son pointeur vers l'arbre.
Enfin, signalons que le symbole nil est une valeur particulière dont la signification eat: "ne pointe vers aucun élément".

Les différents systèmea aeront représentés comme suit (EXPRESSION):

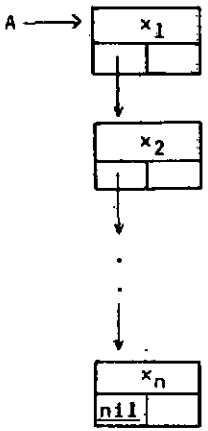
- le syatème aimple: A — x

  aera repréaenté par: A ⟶ x

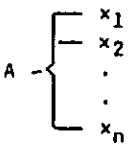
- le syatème-bar:



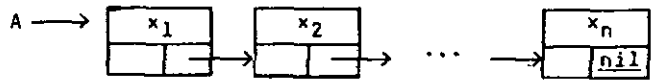
  aera repréaenté par:



- le syatème-bra:

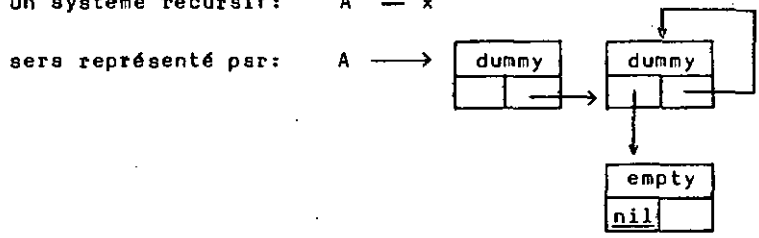


  aera repréaenté par:

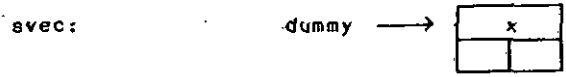


- un système récursif: A = x

sera représenté par:



avec:



Un certain nombre de remarques s'imposent en ce qui concerne un système récursif. On se souvient qu'il est nécessaire de "passer" au moins une fois dans tout système récursif. Ce "passage" se fait en visitant le premier des noeuds contenant le symbole non-terminal "dummy". Le successeur de ce noeud est à nouveau un noeud contenant le symbole "dummy", dont le successeur est: lui-même. Ce qui signifie qu'il sera possible de visiter un nombre quelconque de fois (0,1,2,...) ce second noeud. Dès que la visite de ce noeud ne permet plus d'avancer dans la chaîne à analyser, il convient de choisir une alternative; en l'occurrence, le noeud contenant le symbole spécial "empty". Le noeud "empty" correspond à l'analyse de la chaîne vide, donc, le passage dans ce noeud est toujours possible. Si le successeur de ce noeud est différent de nil, on le suivra pour continuer l'analyse, par contre, s'il est égal à nil, cela signifie qu'on est arrivé à la fin de l'analyse.

Dans la pratique, il est extrêmement rare que x soit un symbole (terminal ou non-terminal). Dans presque tous les cas, il s'agit d'un système-bar ou d'un système composé. Comme un système récursif se réfère toujours deux fois à x, cela implique que l'on construise toujours deux fois le système x.

C'est la raison pour laquelle on a choisi d'associer à tout système récursif un non-terminal "bidon" (GIVENAME) qui est la condition d'entrée du système x et qui permet, par conséquent, de ne construire x qu'une seule fois.

Dans les schémas qui précèdent, on a toujours A ⟶. Cette flèche désigne le pointeur qui va du non-terminal vers le "morceau" de réseau qu'il définit. Ce pointeur est donc contenu dans le symbole lui-même.

Enfin, la différence fondamentale qu'il y a entre un arbre et un graphe (réseau) réside dans la manière de les parcourir. Si, dans le cas d'un arbre, il y a toujours un chemin d'accès unique vers un élément donné, il n'en va pas de même pour un graphe. Comme il est possible d'atteindre le noeud d'un graphe par différents chemins, il est nécessaire de marquer chaque noeud parcouru, ceci afin de ne pas le visiter plusieurs fois et pour éviter de boucler indéfiniment. Par conséquent, on introduit dans chaque noeud du réseau une variable supplémentaire permettant, lors d'un parcours, de savoir si le noeud a déjà été visité ou non. Il s'agit d'un compteur qui est incrémenté de un à chaque parcours et remis à zéro, dans tout le réseau, s'il atteint une certaine valeur limite (INCVISIT). Par référence à la mythologie grecque, on a coutume d'appeler un tel compteur un fil d'Ariane. BARBARA exige que l'on ait, non pas un fil d'Ariane, mais deux. En effet, il est de nombreux cas dans lesquels on doit parcourir le réseau deux fois, simultanément. Par exemple, lors de la destruction d'un "morceau" de réseau (premier parcours) on peut être amené à détruire un système-con ou -any et l'on sait que la destruction d'un tel système est signalée, ce qui nécessite un second parcours.

Les conditions.

Le terme de "condition" désigne soit un système-con soit un système-any. D'une manière générale, une condition associe entre eux un ensemble de noeuds du réseau pour former la condition d'entrée d'un nouveau système. Lorsqu'on introduit dans BARBARA des systèmes simples, -bar ou -bra, les identificateurs désignent toujours des symboles (terminaux ou non-terminaux) tandis

que les identificateurs en paramètres d'une condition, comme dans con(a ENTRY(b) c) = 0, désignent des noeuds appartenant au réseau. Il s'agit-là d'une différence importante. En effet, il n'y a pas obligatoirement bijection entre les symboles et les noeuds, puisque plusieurs noeuds peuvent se référer au même symbole. Deux cas sont à envisager (LOCNODE, LOCSYMB):

- le noeud, en paramètre de la condition, n'apparaît qu'une seule fois dans le réseau.
  Dans ce cas, il y a bijection entre le noeud et le symbole. De manière à pouvoir retrouver le noeud immédiatement il a été décidé de maintenir, dans le symbole, un pointeur supplémentaire qui va du symbole vers le noeud. On a donc un pointeur du noeud vers le symbole et un pointeur du symbole vers le noeud. Il est clair que ce dernier pointeur n'a de sens que si et seulement si le compteur des références aux symboles est égal à un. A noter que la destruction d'une partie du réseau - par exemple lorsqu'on redéfinit un non-terminal - peut nécessiter la mise à jour des pointeurs allant des symboles aux noeuds (SUPTODATE). Ce sera le cas, chaque fois que le compteur des références aux symboles passe d'une valeur différente de un à une valeur égale à un.

- le noeud, en paramètre de la condition, apparaît plusieurs fois dans le réseau.
  Lorsqu'un symbole apparaît plusieurs fois dans le réseau, on a vu qu'il fallait, pour le désigner, le faire précéder de sa condition d'entrée (si tel n'était pas le cas, le programme signalerait une erreur). Les conditions d'entrée sont gérées au moyen d'une pile, également représentée sous forme de liste (PUSH, UNSTACK, STACKISEMPTY, etc...).

  Pour repérer le noeud, on lance une recherche, à partir de cette condition d'entrée, dans l'ensemble du "morceau" de réseau qu'elle définit et dans toutes les "règles" qui lui sont "connectées" (FIND). Dans le cas de l'exemple con(a ENTRY(b) c) = 0, cela reviendrait à rechercher un noeud pointant vers le symbole b dans l'ensemble du sous-réseau dont la condition d'entrée est ENTRY.

Finalement, il nous reste à examiner comment représenter une condition. Chaque noeud, lié à une condition, comprend un pointeur vers un enregistrement qui le décrit. Cet enregistrement contient plusieurs informations:

- un pointeur vers le système que définit cette condition. Dans le cas de l'exemple con(a ENTRY(b) c) = D, un pointeur vers le noeud qui a comme symbole un pointeur vers D.

- le genre de condition à laquelle on a affaire: système-con ou système-any. Dans le cas de l'exemple, il s'agit d'un système-con.

- une liste chaînée des noeuds concernés par la condition. Dans le cas de l'exemple, la liste chaînerait les noeuds: a, ENTRY(b) et c.

- etc...

A noter qu'un noeud ne peut jamais être lié qu'à une seule condition. Chaque fois que l'utilisateur introduit un système-con ou -any, le programme vérifie que les noeuds (en paramètres de la condition) ne sont pas déjà liés à une condition. Le cas échéant, BARBARA signale l'ensemble des noeuds déjà liés (WRITENODE) et imprime le ou les systèmes auxquels ils le sont (WRITECOND). Comme pour la redéfinition d'un non-terminal, le programme se fait confirmer s'il s'agit d'une erreur ou d'une volonté délibérée de redéfinir différemment la condition, auquel cas, il se charge de détruire l'ensemble des systèmes-con ou -any qui doivent l'être.

Connexité.

Un réseau doit nécessairement comporter une condition d'entrée principale. Lorsqu'on lance l'exécution du programme, la première information qu'il demande est précisément le nom de cette condition d'entrée. Toute règle qui n'est pas directement ou indirectement "connectée" à la condition d'entrée principale est dite non connexe. En tout temps, il est possible de demander une liste des règles non connexes (NOTCONNERULES); voir paragraphe

4.7. Lorsque le réseau a été entièrement introduit, BARBARA se charge de signaler l'ensemble des règles non connexes. Elles peuvent avoir été introduites par erreur ou en vue de n'être rendues connexes que lors d'une exécution ultérieure; voir paragraphe 4.6. Dans tous les cas, elles ne seront jamais utilisées, puisque non accessibles. Il convient donc de les détruire (DELNOTCONRULES) et de restituer à la "mémoire libre" l'ensemble des noeuds concernés et l'ensemble des symboles éventuels qui n'apparaissaient que dans ces règles.

Exemple.

Le réseau abstrait de la figure 4.1 serait introduit dans BARBARA par l'ensemble des règles de définition suivantes:

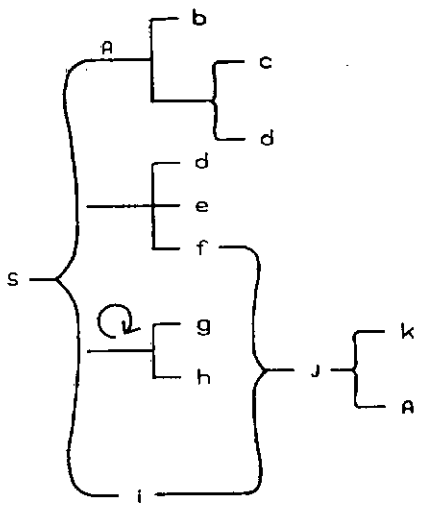
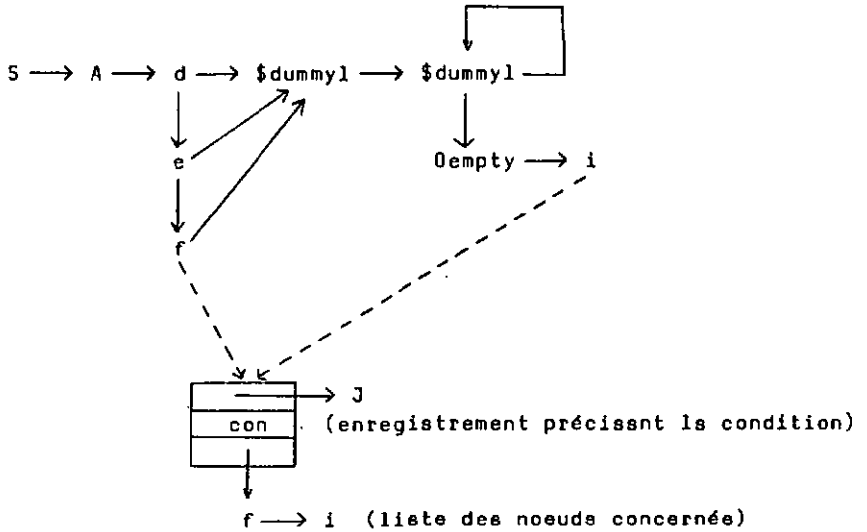
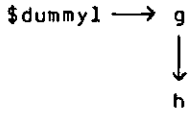


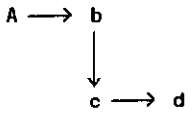
Figure 4.1

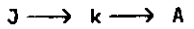
```
S=brs(A bsr(d e f) rec(bsr(g h)) i);
A=bsr(b bre(c d));
con(f i)=J;
J=brs(k A);
```

Il sersit représenté comme suit:









Remarques:

- les flèches situées à droite des identificateurs représentent les successeurs et celles en-dessous, les alternatives. L'absence de flèche remplace le symbole spécial nil.

- les flèches en pointillé représentent les pointeurs vers une condition.

- les non-terminaux "bidon" d'un système récursif s'appellent en fait $dummy1, $dummy2,... etc. De même, le symbole spécial "empty" s'appelle, dans le programme, 0empty (zéro suivi de empty). Ces noms ne satisfont pas la syntaxe d'un identificateur (puisque leur premier caractère n'est pas une lettre), ceci afin de ne pas restreindre la liste des identificateurs possibles, comme empty, dummy1, ... etc.

- les schémas ci-dessus ne sont pas tout à fait corrects, puisqu'aucun identificateur ne devrait y apparaître. On aurait dû dessiner des pointeurs vers les identificateurs qui, eux, sont maintenus ailleurs, dans un arbre de recherche. Cela n'a pas été fait, pour des raisons évidentes de lisibilité.

4.4 Traitement du code (READCODE)

Après l'introduction d'un réseau complet, le programme est en mesure d'accepter le code. Un ensemble d'options choisies dans le réseau, est toujours identifié par ce que nous conviendrons d'appeler une étiquette. Par exemple, à partir de la figure 4.1, on pourrait donner code1 = b e g 1. "code1" est donc l'étiquette, l'expression tout entière s'appelle une réalisation. Une réalisation, comme une règle, se termine toujours par un point-virgule (;).

Lors de l'introduction du code, le programme se livre à une double activité (ANALCODE):

- vérifier que l'ensemble des options choisies est bien une option possible du réseau (PARSE), c'est-à-dire, appartient à l'ensemble des paradigmes.

- mémoriser sur le réseau le chemin emprunté par chacune des réalisation. Dans le cas de l'exemple, il s'agit de créer une relation entre chacun des noeuds b, e, g, i et l'étiquette "code1".

Les étiquettes.

Les étiquettes sont des identificateurs qui sont également organisés dans un arbre de recherche (SEARCHLABEL), avec comme clé, l'identificateur lui-même, c'est-à-dire une chaîne de caractères. Syntaxiquement, une étiquette ne peut jamais être confondue avec un élément de réseau (terminal ou non-terminal). Cependant, BARBARA exige que les identificateurs d'étiquettes soient tous différents de ceux du réseau, ceci afin de pouvoir renseigner (sans ambiguïté) l'utilisateur sur la nature des objets qu'il manipule (voir paragraphe 4.7).

Toute étiquette qui apparaît pour la première fois dans le système est insérée dans l'arbre de recherche, en respectant la relation d'ordre définie sur l'arbre. Si, par contre, on tente d'introduire une étiquette qui figure déjà dans l'arbre, le programme se fait confirmer s'il s'agit d'une erreur ou d'une redéfinition. Dans ce dernier cas, il se charge de détruire, dans tout le réseau, les relations qui existaient entre les noeuds (du réseau) et l'ancienne étiquette (RETNOLAB).

La relation réseau-code.

Toute réalisation doit être mémorisée sur le réseau. Il s'agit donc d'associer à chaque noeud un ensemble d'étiquettes, celles qui ont "passé" par le noeud. Le nombre de réalisations, qui sera introduit, n'est évidemment pas connu a priori. Par ailleurs, PASCAL n'offre pas la possibilité de définir des

ensembles dynamiques. Enfin, la taille d'un ensemble, en PASCAL, est limitée à de petites valeurs: en général, au nombre de bits contenus dans un mot-machine (sur PASCAL CDC, 59 éléments au maximum). Toutes ces raisons nous conduisent à introduire dans les noeuds du réseau, non pas un ensemble d'étiquettes, mais un pointeur vers ce que l'on conviendra d'appeler un ensemble dynamique. Un tel ensemble se constitue d'une liste d'ensembles, dont les éléments sont des entiers compris dans l'intervalle 0..sizeofset. En effet, on rappelle que PASCAL exige que les éléments d'un ensemble soient de type scalaire ou intervalle. Dans notre cas, les éléments sont des identificateurs (les étiquettes) donc de type packed array [1..alfaleng] of char qui n'est ni un type scalaire ni un type intervalle. Cela nous oblige donc à numéroter à partir de zéro chaque étiquette, de manière univoque, et à maintenir ce numéro dans l'arbre de recherche. Inversement, en parcourant l'arbre, il est facile de retrouver une étiquette à partir d'un numéro donné.

L'ensemble dynamique est défini comme suit:

```
type psetoflab = ↑setoflab;
     setoflab  = packed record
                   rset : set of 0..sizeofset;
                   next : psetoflab
                 end;
```

Tant que le nombre d'étiquettes est inférieur à sizeofset + 1, la liste qui constitue l'ensemble n'a qu'un seul élément. Par contre, on devra lui en rajouter un à chaque multiple de sizeofset + 1.

Toutes les instructions de PASCAL, permettant la manipulation d'ensemble, devront être récrites (à l'aide de procédures) pour l'ensemble dynamique (AFFECT, XINSET, NEWSET, UNION, INTER, COMPL, ... etc). Par exemple, on a:

- si x, de type rangelab, est un numéro quelconque d'étiquette, la fonction XINSET est vraie si x appartient à l'ensemble dynamique pointé par p, fausse sinon.

```
function XINSET(x : rangelsb; p : psetofiab) : boolean;
var i : integer;
begin
 i:=x div sizeofsetplusi; (*élément concerné de la liste*)
 (*recherche de cet élément*)
 while i ≠ 0 do begin p:=p↑.next; i:=i-i end;
 xinset:=x mod sizeofsetplusl in p↑.rset
end;
```

- tous les opérateurs sont réalisés de la manière suivante:
  pl:=pl <opérateur> p2.

```
procedure UNiON(pl,p2 : paetofisb);
begin
 repeat
  pi↑.rset:=pl↑.rset + p2↑.raet;
  pl:=pl↑.next; p2:=p2↑.next
 untii pl = nii
end;
```

Comme son nom l'indique, l'ensemble "dynamique" peut donc être adapté au nombre d'étiquettes par pas de sizeofset + 1. Chaque fois que cela sera nécessaire, on étendra l'ensemble dans chacun des noeuds du résesu. Considérons les déclarations suivantes:

a) l'arbre de recherche des symboles:

```
type psymb = ↑symb;
     symb  = packed record
               name          : alfa;
               left,right    : psymb;
               nref          : 0..maxref;
               nodifl        : pnode;
               ptcond        : pcond;
               case terminal : boolean of
                true  : ();
                faise : (entry     : pnode;
                         in        : rangein;
                         connected : boolean)
             end;
```

nama : contient l'identificateur du symbole.
left et right: pointeur vers le sous-arbre gauche, respectivement vers le sous-arbre droit.
nref : compteur des références.
nodifl : pointeur vers le noeud, en cas de bijection. Donc, n'a de sens que si et seulement si: nref = 1.
ptcond : pointeur vers une éventuelle condition. Sera vu au paragraphe 4.7.
entry : pointeur vers le "morceau" de réseau que définit le non-terminal.
ln : permet de garder la ligne concernée du fichier. Sera vu au paragraphe 4.6.
connected : vrai si la règle est connexe.

b) le réseau:

```
type pnode = ↑node;
     node  = packed record
               ego     : psymb;
               alt,suc : pnode;
               pset    : psetoflab;
               visit   : packed array [mark] of rangevisit;
               case condition : boolean of
                 false : ();
                 true  : (which : pcond)
             end;
```

ego : pointeur vers le symbole.
alt et suc: pointeur vers l'alternative, respectivement le successeur.
pset : pointeur vers l'ensemble dynamique des étiquettes.
visit : fil d'Ariane. Il s'agit en fait d'un tableau de fils d'Ariane, ne sachant pas, a priori, combien de fils seraient nécessaires. En l'occurrence, il n'y en a que deux (l'un appelé al, l'autre we) donc mark = (al,we).

```
which      : pointeur  vers  l'enregistrement précisent la condi-
             tion
```

c) les conditions:

```
type pcond      = ↑cond;
     cond       = packed record
                    condentry : pnode;
                    kind      : consy..anysy;
                    concerned : plistnode;
                    written   : rangevisit;
                    ln        : rangeln
                  end;
     plistnode = ↑listnode;
     listnode  = packed record
                    who       : pnode;
                    ancestor  : psymb;
                    next      : plistnode
                  end;
```

condentry: pointeur vers le système que définit la condition.
kind : genre de condition: système-con ou -any. Il s'agit d'un intervalle faisant partie d'un type scalaire plus général: symbol.
concerned: pointeur vers la liste des noeuds concernés.
written : permet, lorsqu'on écrit un ensemble de noeuds sppartenant déjà à une condition, de l'écrire en les regroupant.
ln : permet de garder la ligne concernée du fichier. Sera vu en 4.6.
who : pointeur vers le noeud concerné par la condition.
ancestor : pointeur vers le symbole PERE (au sens du parapraphe 2.3).
next : pointeur vers l'élément suivant de la liste.

On considère une première procédure, tout à fait générale, PASSGRAPH, qui permet d'effectuer une sction quelconque (passée en paramètre) sur l'ensemble de tous les noeuds du réseau. Elle visite l'arbre des symboles et pour chsque non-terminal qu'elle

rencontre, elle exécute une action sur le "morceau" de réseau qu'il définit.

```
procedure PASSGRAPH(p : paymb; procedure action);
begin
 if p ≠ nil then
 with p↑ do
 begin
  if not terminal then action(entry);
  passgraph(left);  (*visite du sous-arbre gauche*)
  passgraph(right)  (*visite du sous-arbre droit*)
 end
end;
```

Dès lors, l'extension de l'ensemble dynamique pourra se faire grâce à la procédure INCREASESET, passée en paramètre de PASSGRAPH.

```
procedure INCREASESET(p : pnode);
var q,r : psetofisb;
begin
 if p ≠ nil then
 with p↑ do
 if visit[si] ≠ cv[si] then (*noeud pas encore visité*)
 begin (*recherche  du dernier élément de la liste constituant*)
       (*l'ensemble dynamique*)
  q:=pset;
  while q↑.next ≠ nil do q:=q↑.next;
  (*fabrication d'un nouvel élément*)
  new(r); r↑.rset:=[]; r↑.next:=nil;
  q↑.next:=r; (*on l'accroche à la liste*)
  (*parcours d'un éventuel système défini par une condition*)
  if condition then increaseset(which↑.condentry);
  increaseset(alt); (*parcours de l'alternative*)
  increaseset(suc)  (*parcours du successeur*)
 end
end;
```

L'analyse du code.

Sans entrer dans les détails de programmation, il est possible de décrire ce qu'est la procédure PARSE, qui se charge de l'analyse du code. "goal" est le pointeur qui parcourt le réseau. "p" est un pointeur vers le noeud qui correspond au symbole courant de la chaîne à analyser. Il est fourni par la procédure LOCNODE et lorsqu'on entre dans PARSE, il pointe vers le noeud correspondant au premier symbole de la chaîne. "match" est vrai chaque fois que les symboles du réseau et de la chaîne correspondent.

```
procedure PARSE(goal : pnode; var match : boolean);
var isconsa : boolean;(*vrai si toutes les conditions d'entrée*)
                      (*d'un aystème-con sont setisfaites*)
    pln     : plistnode; (*variable de service*)
begin
 repeat
  with goal↑,ego↑ do
  begin
   if terminal then
    if p = goal then
     begin match:=true; locnode(p) end
    else match:= name = empty
   else (*non-terminal*) parse(entry,match);
   if match then
    begin
     visit[we]:=cv[we]; (*marque le noeud*)
     (*manque le détail du code permettant d'insérer le numéro*)
     (*courant d'étiquette dans l'ensemble dynamique*)
     if condition then
     begin
      with which↑ do
      begin
       if kind = anysy then parse(condentry,match)
       else begin
        (*système-con. Donc, vérifier que toutes les condi-*)
        (*tions d'entrée sont satisfaites*)
        pln:=concerned;
        repeat
```

```
          isconsa:=pln↑.who↑.visit[we] = cv[we];
          pln:=pln↑.next
        until (pln = nil) or not isconsa;
        if isconsa then parse(condentry,match)
       end (*else*)
      end (*with*)
     end; (*condition*)
     gosl:=suc (*comme "match", on prend le sucesseur*)
    end (*match*)
   else gosl:=slt (*comme non "match" on prend l'alternative*)
  end
 until gosl = nil
end;
```

La procédure PARSE est toujours appelée en substituant à "gosl" la condition d'entrée principale du réseau. Une chaîne est acceptée si, en sortant de PARSE, on a "match" vrai et que l'analyse a conduit au point-virgule, terminateur de toute réalisation.

4.5 Exploitation de l'ensemble réseau-code (QUESTIONING)

Il n'y a que fort peu de chose à dire sur cette troisième et dernière phase du programme BARBARA. A ce stade, on dispose donc d'un réseau dont chaque noeud est muni d'un ensemble qui indique les divers chemins empruntés par les réalisations. Le programme est capable de calculer n'importe quelle expression de type ensembliste (EXPRES).

Les expressions.

Les opérandes d'une expression sont les noeuds du réseau, sous-entendu, les ensembles dynamiques reliés aux noeuds. Il y a deux opérateurs binaires or et and et un opérateur unaire not. On dispose également des parenthèses gauche et droite, permettant la mise en évidence d'une sous-expression.

Soit R, la condition d'entrée principale d'un réseau et soit A, B, C trois noeuds quelconques de ce réseau. Alors:

- A or B signifie A ∪ B (UNION).

- A and B signifie A ∩ B (INTER).

- not A signifie R - A, c'est-à-dire l'ensemble des réalisations qui ont la propriété R (donc toutes) à l'exclusion de celles qui ont la propriété A (COMPL).

A noter que l'opérateur and est prioritaire sur l'opérateur or. Donc, A or B and C est évalué A or (B and C).

Le résultat d'une expression est évidemment un ensemble dont on peut connaître, grâce à la fonction LIST, quelles sont les étiquettes qui le constituent (PRINTRESULT) ou, grâce à la fonction NUM, quel est son cardinal (POWERSET).

Les tableaux croisés.

BARBARA offre la possibilité d'effectuer des tableaux croisés, grâce à la fonction TAB (la syntaxe précise de cette fonction, de même que celle des fonctions LIST et NUM, sera vue au chapitre 5) (CROSSTABULATION). Un tableau croisé se définit à partir de deux ensembles de noeuds. Le premier ensemble comprend les éléments formant les colonnes du tableau, le second, les lignes. Chaque noeud lu est introduit dans une liste chaînée, celle des colonnes, respectivement celle des lignes (ADDELEM). Puis le programme calcule le cardinal de l'intersection de chacun des éléments de la première liste avec chaque élément de la seconde.

Si le deuxième ensemble (celui des lignes) peut comporter un nombre quelconque d'éléments, il n'en va pas de même pour le premier. En effet, toute sortie effectuée par BARBARA le sera sur un certain support physique, tel qu'une imprimante, un terminal à écran ou à papier, etc... mais dans tous les cas, limité dans sa largeur. Donc, d'une manière générale, le programme doit tenir à jour un compteur de caractères qui, s'il

atteint une certaine valeur limite, force un passage à la ligne (REMAINS). L'ensemble des noeuds en colonne est donc limité en fonction du nombre et de la longueur des identificateurs qui les désignent.

Enfin, signalons l'existence de la fonction TER à un seul argument, disons X, un non-terminal quelconque. Cette fonction se charge de calculer l'ensemble des éléments terminaux accessibles à partir de X. Pour ce faire, elle parcourt tout le sous-réseau de condition d'entrée X et introduit dans la liste des noeuds, paramètres de la fonction TAB, tout terminal rencontré (ADDTERMINAL).

4.6 Conservation des données sur fichiers

Lorsqu'on introduit dans BARBARA des règles ou des réalisations, on désire, dans la plupart des cas, les rendre permanentes, c'est-à-dire leur donner une durée de vie "infinie". Ceci se fait en les conservant dans des fichiers.

Il est clair qu'à un réseau donné, ne peut correspondre qu'un code bien particulier: il ne doit contenir que des choix possibles dans ce réseau. Il a donc été décidé de conserver le réseau et le code dans un même fichier, qui est constitué, cependant, de deux parties distinctes, ceci afin de pouvoir modifier l'une sans se préoccuper de l'autre. Un fichier peut être "partitionné" physiquement. Ce sera le cas sur la série des ordinateurs CDC (systèmes d'exploitation NOS, SCOPE, NOS/BE, etc...) où il est possible d'avoir ce que l'on appelle des fichiers segmentés. Sur une machine ne possédant pas cette caractéristique, on procèdera à une séparation logique en marquant la fin de chaque partie à l'aide d'un symbole spécial et unique. Dans tous les cas, on conviendra d'appeler segment chacune des parties du fichier.

Le programme BARBARA a l'en-tête suivant:

```
program BARBARA(data,input,output).
```

"data" est le fichier des données (permanentes); "input" correspond au "clavier" du terminal à partir duquel on exécute BARBARA et "output" correspond à l'"écran" de ce même terminal.

Le réseau.

Deux cas sont à envisager:

- le segment-réseau est vide.
  Le programme invite alors l'utilisateur à introduire un réseau, en lui donnant la possibilité de le rendre permanent ou non. Pour être rendue permanente, une règle doit être correcte tant du point de vue syntaxique que du point de vue sémantique. Donc, tout caractère lu sur "input" est accumulé dans un tampon. Si l'ensemble des caractères forme une règle correcte, le tampon est copié sur un fichier interne temporaire, organisé en lignes (COPYBUFF), sinon, il est réinitialisé en vue d'accepter une nouvelle donnée.

  Chaque règle introduite commence donc à une certaine ligne du fichier temporaire et se termine nécessairement par un point-virgule. On garde dans tout symbole non-terminal le numéro de la ligne à laquelle commence le "morceau" de réseau qu'il définit (il s'agit de l'identificateur "ln" vu au paragraphe 4.4, lettre a). Par ailleurs, on a vu qu'il était possible, après avoir donné une certaine définition à un non-terminal, d'en donner une définition différente. Cela nécessite non seulement de corriger dans le symbole la valeur de "ln" mais également de "désactiver" l'ensemble des lignes où apparaissait la première définition. Cette "désactivation" se fait en tenant à jour une liste chaînée des lignes qui devront être annulées (*INITERLIST* et INSERTERLN). Ceci explique qu'on ait été obligé de passer par un fichier temporaire. Ce n'est qu'à la fin de la définition du réseau que le fichier temporaire sera "rembobiné" pour être ensuite recopié sur le fichier "data", en omettant certaines parties, conformément à la liste chaînée des lignes "désactivées" (COPYTEDA).

- le segment-réseau est non vide.
  Le programme lit alors l'ensemble de l'information qui se trouve sur le segment et construit le réseau qui lui correspond; toute règle non connexe est évidemment signalée. Après cette lecture, l'utilisateur peut décider de modifier ou non le réseau et, s'il y a modifications, de les rendre permanentes ou non.

  En cas de modifications permanentes, le fichier "data" est recopié sur le fichier temporaire et les éventuelles règles non connexes sont supprimées ou non, selon le désir de l'utilisateur. Finalement, on est ramené au même cas que celui du "segment-réseau vide".

Le code.

Si le segment-code est vide, on procède exactement de la même manière que pour le segment-réseau vide. Par contre, s'il n'est pas vide, on est confronté à une difficulté supplémentaire. Supposons que, lors de la première phase du programme, le réseau ait été modifié. Alors une partie ou le code tout entier contenu dans le segment, risque de ne plus correspondre au réseau. Toutes les réalisations incorrectes devront donc être rejetées mais pour pouvoir les corriger, l'utilisateur a besoin qu'on les lui affiche. Cependant, la procédure qui écrit une réalisation (cf. paragraphe 4.7) ne peut le faire qu'à partir des chemins que cette dernière emprunte dans le réseau, chemins qui n'existent évidemment pas si la réalisation est incorrecte.

Il n'y a que le fichier "data" qui garde une trace des réalisations incorrectes, où elles y figurent d'ailleurs réparties de manière aléatoire. Donc, il aurait été nécessaire d'organiser "data" en accès direct, pour pouvoir les retrouver immédiatement. Malheureusement, PASCAL n'offre pas la possibilité de traiter des fichiers en accès direct. Il a donc fallu "simuler" ce type d'accès sur fichier séquentiel (SKIPLNONDA).

Enfin, signalons que les réalisations incorrectes sont traitées de manière analogue aux règles non connexes: possibilité de les supprimer du fichier en cas de modifications permanentes du

code, possibilité de les redéfinir, d'où introduction dans la liste des lignes "désactivées", etc...

4.7 La fonction HELP

BARBARA, on l'a vu, est un programme entièrement interactif, c'est-à-dire, qui se déroule sous forme de dialogue entre l'utilisateur et l'ordinateur. Qui dit dialogue, dit échange d'informations. Le programme doit donc être capable de fournir en tout temps un certain nombre de renseignements sur l'ensemble des objets qu'il manipule, la signification de chacun d'eux, l'état courant du système, etc.... On conviendra de regrouper sous le nom général de HELP tous les mécanismes permettant de réaliser ces diverses fonctions.

Le réseau.

Pour définir un réseau, on est conduit à introduire dans le système un certain nombre d'identificateurs, qui peuvent désigner soit des terminaux soit des non-terminaux. Tout appel à la fonction HELP est marqué par un point d'interrogation (?). Par exemple, pour avoir des renseignements sur un identificateur donné, il suffit de le faire précéder d'un point d'interrogation. BARBARA procède alors à une recherche dans l'arbre où sont maintenus les symboles. Quatre cas peuvent se présenter:

a) la recherche s'avère infructueuse, le symbole est donc "inconnu" du système.

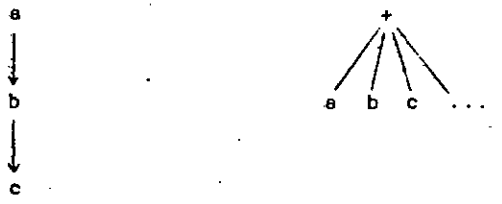
b) il s'agit d'un symbole terminal, dont on peut dire, grâce au compteur des références, combien de fois il apparaît dans le système.

c) il s'agit d'un non-terminal, qui est la condition d'entrée d'un système. Dans ce cas, ce qui intéresse l'utilisateur, c'est de savoir comment il a été défini (WRITEXPR). La seule manière de connaître cette définition consiste à réaliser la fonction inverse de celle qui permet de construire le réseau.

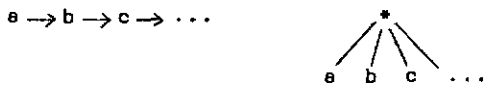
Pour définir un "morceau" de réseau, on dispose essentiellement de deux opérateurs: bar, que l'on représentera par + et bra, que l'on représentera par * (l'opérateur rec n'utilise que des constructions bar et bra et peut être facilement repéré grâce à ses non-terminaux "bidon": $dummy). Enfin, on rappelle que les opérateurs bar et bra sont des opérateurs n-aires. Le problème consiste donc, à partir d'un sous-graphe, à construire l'arbre n-aire de l'expression qui le définit (TEXPRESSION). Il s'agit d'un algorithme relativement complexe que l'on illustrera à l'aide de quelques exemples:

au sous-graphe: correspond l'arbre:

1)

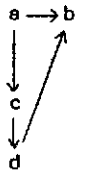


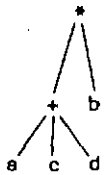
2)



3)

4)





On peut aisément se convaincre de la difficulté de l'algorithme grâce aux exemples 3 et 4. En effet, la seule différence existant entre les deux sous-graphes réside dans le fait qu'il est possible, dans 4, d'atteindre b à partir de d. Cependant, tant que l'on n'est pas parvenu à l'élément d, les arbres construits seront identiques dans les deux cas. D'une manière plus générale, on peut dire que tout circuit dans un sous-graphe (CLOSEO) oblige à une réorganisation complète de l'arbre: permutation d'opérateurs, translation de certains noeuds, etc.... Cet algorithme n'est guère transparent: il procède essentiellement d'une démarche de nature empirique, dont on peut se rendre compte en observant les différences entre les arbres 3 et 4.

Du point de vue de la programmation, la représentation d'un arbre n-aire est particulièrement peu commode. Par contre, il est toujours trivial de donner une représentation binaire d'un arbre n-aire. C'est évidemment l'option qui a été choisie dans BARBARA.

Finalement, une fois l'arbre construit, en le parcourant de manière pré-ordonnée, gauche - droite, on peut reproduire l'expression qu'il définit.

d) il s'agit d'un symbole (terminal ou non) qui définit un système-con ou -any, comme ce serait le cas de d, dans con(a b c) = d. On se souvient qu'il existe, dans le symbole, un pointeur vers l'enregistrement qui précise la condition (il s'agit de l'identificateur "ptcond" vu au paragraphe 4.4, lettre a). On commencera donc par écrire la condition (WRITECOND) puis on traitera normalement le symbole, selon qu'il s'agit d'un terminal ou d'un non-terminal.

Lorsqu'on introduit un réseau, il est indispensable de savoir, en tout temps, s'il y a des règles non connexes. Les fonctions LIST de HELP (donc ?LIST) et NUM (donc ?NUM) donnent la liste, respectivement le nombre des non-terminaux qui seraient la condition d'entrée d'un système non connexe (NOTCONNERULES). BARBARA procède de la manière suivante:

- en parcourant l'arbre de recherche des symboles, il initialise tout non-terminal rencontré comme s'il était non connexe (voir identificateur "connected", paragraphe 4.4, lettre a) (INIT-CONNE)

- à partir de la condition d'entrée principale du réseau, il parcourt le graphe et l'ensemble de tous les sous-graphes possibles en marquant tout non-terminal rencontré comme connexe (CONNE)

- un deuxième parcours de l'arbre permet d'écrire, respectivement de compter, les non-terminaux non connexes (NMCONNE).

Le code.

? <identificateur d'étiquette> donne la liste des options choisies dans le réseau pour l'étiquette concernée. En parcourant l'arbre de recherche des étiquettes, on peut rencontrer l'un des trois cas suivants:

- recherche infructueuse, donc étiquette "inconnue"

- on trouve une étiquette "incorrecte", c'est-à-dire qui ne correspond plus au réseau actuellement défini. BARBARA la recherche alors dans le fichier "data" comme expliqué au paragraphe 4.6

- l'étiquette est "correcte", le programme parcourt alors le réseau et écrit tout noeud qui contient cette étiquette. A noter que seuls les noeuds pointant vers un symbole terminal sont écrits. S'il s'agit d'un symbole auquel on se réfère plusieurs fois, on le fera précéder de sa condition d'entrée pour soulever l'ambiguïté. La réalisation n'est donc pas

reproduite exactement comme elle a été introduite (mise en pages, utilisation de certains non-terminaux visant à accroître la lisibilité, etc...), il s'agit plutôt d'un "aide-mémoire".

Lors de l'introduction du code, ?LIST et ?NUM donnent la liste, respectivement le nombre des étiquettes "incorrectes", c'est-à-dire qui définissent des options non conformes au réseau présent en mémoire. Ceci se fait en parcourant l'arbre de recherche des étiquettes et en écrivant, respectivement en comptant, toutes celles marquées comme "incorrectes", information donnée par un booléen maintenu dans chaque élément de l'arbre.

4.8 Traitement des erreurs

Tout programme qui travaille en mode "batch", c'est-à-dire auquel on fournit d'emblée l'ensemble de toutes les données utiles à son fonctionnement, nécessite un traitement des erreurs relativement sophistiqué - la plupart des compilateurs de langages de programmation travaillent dans ce mode -. En effet, il n'est guère concevable qu'un compilateur cesse son activité sitôt la première erreur rencontrée: il doit être capable de poursuivre son travail de vérification jusqu'à la fin du programme qu'il compile, en signalant le maximum d'erreurs possibles. Pour ce faire, il doit ignorer certaines données et tenter de se "rattraper" dans l'analyse du code source. L'ensemble des mécanismes permettant de réaliser ces diverses fonctions est connu sous le nom de recouvrement d'erreurs (error recovery).

Dans le cas d'un programme en mode "interactif", les choses sont évidemment considérablement simplifiées. En effet, comme les données sont entrées au fur et à mesure des besoins (en fait, ligne à ligne), il est possible, en cas d'erreur, de réagir immédiatemment.

Toutes les procédures de BARBARA susceptibles de détecter une erreur ont un point de reprise (une étiquette au sens de PASCAL)

qui leur permet, en cas d'erreur, de recommencer une nouvelle analyse. En outre, elles possèdent toutes une procédure locale ERROR1, ERROR2, ... etc dont l'activité consiste, au moins:

- à mettre un indicateur d'erreur à vrai

- à appeler une procédure globale (ERROR) qui édite un message et "saute" l'ensemble des caractères depuis l'endroit où l'erreur a été détectée jusqu'au prochain point-virgule ou jusqu'à la fin de la ligne qui venait d'être introduite

- à se brancher au point de reprise adéquat.

Si, au point de reprise, l'indicateur d'erreur est en fonction, le programme se charge de détruire l'ensemble des noeuds qui ont déjà pu être construits, si l'on était en train de définir un réseau (RETNET), ou l'ensemble des étiquettes qui sont déjà introduites dans le réseau, si l'on était en train de définir le code (RETNDLAB).

4.9 Limites et perspectives

Limites.

On a vu que BARBARA offre la possibilité de modifier très aisément le réseau. Cependant, toute modification du réseau se répercute nécessairement sur le code. Dans sa version actuelle, le programme réagit simplement en signalant combien de réalisations ne correspondent plus au réseau et pour quelles raisons. Par ailleurs, il supprime entièrement toute réalisation erronée, c'est-à-dire qu'il la considère comme inexistante. Cette façon de procéder est assez grossière puisqu'elle oblige l'utilisateur à réintroduire entièrement toute la réalisation, alors que, dans la plupart des cas, seule une partie de la réalisation nécessite des modifications.

Dans le cas de la suppression d'une partie de réseau, on pourrait imaginer que le programme enlève lui-même l'ensemble du

code qui correspondrait à la partie manquante du réseau. Inversement, lorsque l'utilisateur décide de développer le réseau, c'est-à-dire de considérer un non-terminal comme la condition d'entrée d'un nouveau système, il serait agréable de ne pas devoir réintroduire les réalisations toutes entières mais uniquement la partie correspondant au nouveau système.

Pour qu'une réalisation soit correcte, il est nécessaire que tous ses constituants correspondent exactement au réseau, tant du point de vue de leur nom que du point de vue de leur place dans la réalisation. Ceci revient en fait à considérer toute réalisation comme une copie partielle (au sens large) du réseau. En fait, on pourrait imaginer des mécanismes beaucoup plus subtils. Par exemple, si l'on désirait rendre le code plus proche de la langue naturelle, il serait nécessaire de pouvoir procéder à certaines transformations des réalisations.

Supposons que l'on considère certains termes comme des substantifs, on pourrait envisager de donner des règles de transformation permettant de marquer le singulier ou le pluriel. Un autre cas consisterait à considérer le terme comme la racine d'un verbe et à donner des règles de transformation permettant la conjugaison de ce verbe. Dans tous les cas, les termes de la réalisation ne correspondraient plus à ceux du réseau.

Le système-bra impose une place bien définie à chacun des termes de la réalisation. En effet, si l'on considère le sytème x = bra(x_1 x_2 ... x_n) et qu'une réalisation satisfait la condition d'entrée x alors x_1 doit précéder x_2, etc... qui doit précéder x_n. Là aussi, on pourrait imaginer vouloir définir un ordre différent. Bien que l'ordre des constituants d'une réalisation soit fixé par avance, le code est en fait considéré par le programme comme un ensemble non structuré. Ceci pose un problème qu'il est facile de comprendre grâce à un exemple. Reprenons une partie du réseau de la figure 3.7 que l'on a traité en détail au paragraphe 3.3:

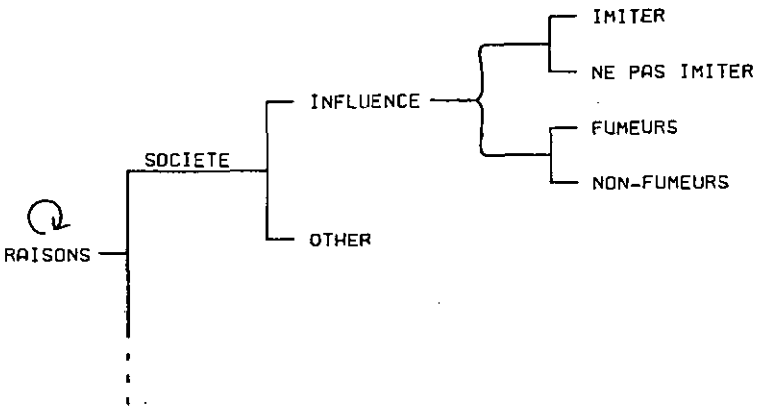


Figure 4.2

Supposons que l'on ait à coder une phrase du genre: "j'ai voulu imiter les non-fumeurs, c'est-à-dire ne pas imiter les fumeurs" (un tel cas se produit dans la réponse fournie par B). Formellement on a:

```
B = RAISONS(SOCIETE(imiter non-fumeurs))
    RAISONS(SOCIETE(ne pas imiter fumeurs))
```

et le programme mémorise les quatre caractéristiques: imiter, ne pas imiter, fumeurs, non-fumeurs. Dès lors, il n'est plus possible de savoir si B a: "imiter les fumeurs" ou "imiter les non-fumeurs", ce qui évidemment n'est pas la même chose! Pour éviter de telles contradictions, qui, notons-le ne peuvent survenir que dans le cas d'un système récursif, on aimerait pouvoir lier entre eux ou exclure mutuellement certains termes.

On a vu que l'un des avantages de la méthode proposée est qu'elle permet de procéder par raffinements successifs, ce qui signifie que tout élément terminal du réseau peut être considéré comme une condition d'entrée potentielle d'un nouveau système. Ou, dit autrement, qu'un réseau peut être considéré comme étant constitué lui-même d'un ensemble de réseaux.

Lorsque l'on conçoit un réseau, on procède toujours par raffinements successifs. Par contre, le programme est relativement mal adapté à cette démarche. En effet, il ne considère toujours qu'un seul réseau et un seul code. Pour faciliter une conception par raffinements successifs, il serait préférable de disposer de plusieurs réseaux auxquels correspondraient plusieurs codes et de prévoir des mécanismes permettant de combiner ces différents réseaux; le programme se chargeant de rassembler les codes leur correspondant.

Dès lors, il serait possible d'effectuer une analyse du code selon différents niveaux de finesse, c'est-à-dire de procéder par raffinements successifs non seulement dans la conception du réseau mais également au niveau de l'analyse. A noter qu'un tel outil s'avérerait particulièrement utile lorsque les réalisations sont longues puisqu'il permettrait de les entrer dans le système par fragments.

Perspectives.

Au paragraphe 4.1, il a été mentionné que BARBARA fonctionne comme un compilateur de compilateurs. Il serait donc très facile de modifier légèrement le programme pour en faire un outil permettant la compilation interactive d'un langage de programmation. Il suffit pour cela de donner un réseau qui décrit la syntaxe du langage, le code étant le programme que l'on désire éditer. Dès lors, BARBARA vérifierait immédiatement la syntaxe de toute ligne introduite et serait en mesure de signaler toute erreur de manière précise et détaillée - ce qui est rarement le cas pour les compilateurs usuels - puisqu'il possède en mémoire l'arbre syntaxique complet du langage.

Par ailleurs, la fonction HELP s'avérerait fort utile dans une phase d'apprentissage, puisqu'elle permettrait de savoir en tout temps quelle est la syntaxe du langage de programmation. Enfin, le fait de donner la grammaire sous forme de réseau, outre que cela permet d'avoir un même programme pour plusieurs langages différents, offre un avantage supplémentaire. Il est parfois dangereux, lorsqu'on enseigne un langage de programmation, de donner d'emblée toute sa puissance d'expression; cela conduit

l'étudiant à utiliser certains éléments du langage à mauvais escient et à obscurcir par conséquent l'algorithme. On peut donc parfaitement imaginer que les éléments du langage ne seraient mis à disposition que lorsqu'ils ont été introduits et discutés. A noter qu'il ne manque en gros que la possibilité d'effectuer des vérifications sémantiques pour que BARBARA puisse fonctionner comme on vient de le décrire.

Une autre caractéristique du programme, discutée en 4.2, est qu'il vérifie si la grammaire représentée par le réseau est ou non LL(1). Ce fait, qui a déjà été utilisé, peut être fort utile pour quelqu'un qui conçoit un langage de programmation.

Dans sa phase de conception, un langage est sans cesse modifié. BARBARA semble être parfaitement adéquat pour garder toujours à jour la grammaire dans son état actuel et pour tester les éventuelles incohérences, en particulier la propriété LL(1).

Il est bien connu que la plus grande partie d'un programme d'application est en général consacrée à l'introduction et à la vérification des données. Il s'agit là d'opérations fastidieuses et qui sont toujours de même nature. On souhaiterait résoudre une fois pour toute ce genre de problème à l'aide d'un programme général. Le formalisme des réseaux semble constituer une base adéquate à la solution du problème. En effet, on pourrait parfaitement décrire les données de chaque application particulière à l'aide d'un réseau, ceci d'autant plus que le système-con offrirait la possibilité de poser des conditions sur certaines données, dans le but, par exemple, d'effectuer des tests de cohérence. Moyennant qu'on imagine un formalisme pour la vérification des contraintes, BARBARA semble là aussi offrir de nouvelles possibilités particulièrement intéressantes.

Bien que BARBARA ait déjà été utilisé dans de nombreux cas, il s'agit encore d'un programme relativement jeune et qui, par conséquent, est en constante évolution. La description rapportée dans ce chapitre correspond aux objectifs minimaux que l'on s'était fixé. On peut signaler certains développements déjà réalisés ou en cours de réalisation:

- introduction d'un mini-langage de commande permettant de préciser d'emblée le mode d'exploitation du programme. Par exemple, lorsqu'on ne veut que procéder à l'interrogation de l'ensemble réseau-code, il est gênant de devoir répondre par oui ou non à toutes une série de questions concernant les fichiers ou d'éventuelles modifications de données.

- possibilité d'indicer les éléments du réseau. Par exemple, pour ne pas devoir écrire bar(x1 x2 ... xn) il suffit de dire bar(x[1..n]).

- possibilité de définir des expressions logiques. Lors de l'interrogation des données, il est de nombreux cas dans lesquels certaines expressions logiques reviennent continuellement. Il est donc utile de pouvoir les définir une fois pour toute et de s'y référer à l'aide d'un identificateur. Par ailleurs, il est possible d'utiliser ces expressions prédéfinies comme paramètres des tableaux croisés, ce qui en étend considérablement la puissance.

Mesures d'efficacité.

Dans la plupart des cas traités (des réseaux de quelques dizaines de noeuds et un code de moins de cent réalisations) les temps de réponse sont pratiquement négligeables. On a cependant procédé à un certain nombre de mesures sur un ordinateur VAX 11/780, muni d'un système d'exploitation à mémoire virtuelle.

Il ne faut que quelques secondes pour introduire et constituer un réseau d'environ 75 noeuds. En moins d'une minute, on peut introduire et vérifier sur un tel réseau jusqu'à 2000 réalisations. Dans la phase d'interrogation, l'évaluation d'une expression quelconque est immédiate.

Une autre expérience a consisté à introduire plus de 300 noeuds avec un code de plus de 2000 réalisations. La constitution du réseau a pris environ vingt secondes et celle du code de deux à trois minutes, ce qui est parfaitement raisonnable, étant donné qu'il s'agit de lire un fichier d'environ 2500 lignes. L'évaluation d'une expression quelconque pour interroger ces données

est nettement inférieure à une seconde.

On pourrait théoriquement dépasser ces limites, ce qui, évidemment, n'a aucun sens, étant donné la nature et le genre des informations traitées par BARBARA.

REFERENCES

1. BAUER, F.L. et EICKEL, J. (Eds), Compiler Construction, An Advanced Course. Lectures Notes in Computer Science Vol. 21. Berlin, Springer-Verlag, 1974.

GRIES, D., Compiler Construction for Digital Computers. New York, Wiley, 1971.

2. AHO, A.V. et ULLMANN, J.D., Principles of Compiler Design. Reading (Mass.), Wesley, 1977.

3. KNUTH, D.E., Top-down Syntax Analysis. Acta Informatica, 1971, 1, 79-110.

LEWIS, P.M. et STEARNS, R.E., Syntax-directed Transduction. J. ACM, 1968, 15, 464-488.

4. WIRTH, N., The Design of a PASCAL compiler. Software - Practice and Experience, 1971, 1, 309-333.

5. JENSEN, K. et WIRTH, N., PASCAL - User Manuel and Report. Lectures Notes in Computer Science Vol. 18. Berlin, Springer-Verlag, 1974.

6. COHEN, D.J. et GOTLIEB, C.C., A List Structure Form of Grammars for Syntactic Analysis. Comp. Surveys, 1970, 2, 65-82.

WIRTH, N., Algorithms + Data Structures = Programs. Englewood Cliffs (New Jersey), Prentice-Hall, 1976.

CHAPITRE CINQUIEME

COMMENT UTILISER BARBARA

Ce chspitre contient un mode d'emploi du programme BARBARA. L'aspect syntaxique y est décrit à l'aide de diagrammes. Toutes les situstions que l'on peut rencontrer en cours de dialogue sont examinées et illustrées psr dea exemples.

C'est à dessein que l'on y trouvera parfois certaines explications ou définitions déjà vues précédemment. Ce chapitre constitue un tout, car il est également destiné à d'éventuels utilissteurs de BARBARA qui, dans la plupsrt des cas, ne a'intéressent qu'aux fonctions réalisées psr le programme.

5.1 Remarques générales

Le programme BARBARA, on l'a vu, a été entièrement écrit en PASCAL. Afin d'en faciliter l'sdaptstion sur le plus grsnd nombre d'ordinateurs différents, il n'utilise, en principe, que ce que l'on a coutume d'sppeler du PASCAL Full Standard. Malheureusement, la standardisation n'est jamais parfaite. En particulier:

- le jeu des csractères disponibles est très souvent différent d'un ordinateur à l'autre. Dans les exemples qui suivent, plus préciaément dans les dialogues entre le programme et l'utilisateur, les minuscules désignent les données introduites par l'utilisateur tandis que les majuscules désignent les réponses fournies par le programme. Il s'agit là d'une convention purement typographique qui ne permet pas de déduire que l'on doive disposer dea deux jeux de caractères, majuscules et minuscules.

- les entrées-sorties ne sont que rarement compatibles. Certains compilsteurs PASCAL exigent que le premier caractère de chaque ligne soit un caractère de contrôle, d'autres ne l'exigent pas. Psr ailleurs, la manière de transmettre un fichier su programme varie fort d'une installation à l'sutre. Quant à la gestion des fichiers, (durée de vie, sauvetage, etc..), elle

dépend non seulement du système d'exploitation mais également de décisions de nature purement administrative. Il n'y sera donc jamais fait allusion dans la suite de ce chapitre.

En résumé, les choix d'implantation sont souvent dictés par la machine sur laquelle un programme est développé, en l'occurrence une CDC de la série CYBER. Ceci explique, entre autres, certaines valeurs données aux constantes du programme, le fait qu'il n'a pas été possible de rendre "transparent" à l'utilisateur la manipulation des fichiers, car il aurait fallu disposer pour cela d'"assignations dynamiques de fichier", etc...

Chaque fois que le programme est en mesure d'accepter une donnée, il le signale en affichant un symbole spécial, appelé parfois prompt symbol, et qui, en l'occurrence, est le caractère ">".

La syntaxe des données fournies à BARBARA est précisée à l'aide de diagrammes syntaxiques. Un chemin à travers un tel diagramme définit une donnée syntaxiquement correcte. On distingue deux types de "boîtes":

- des "boîtes" arrondies, qui contiennent toujours des symboles terminaux, c'est-à-dire, des symboles appartenant au vocabulaire de BARBARA.

- des "boîtes" rectangulaires, qui contiennent un nom qui est une référence à un autre diagramme; on retrouve ce nom en titre du diagramme correspondant.

5.2 Exemple liminaire

Considérons le réseau de la figure 5.1. Un tel réseau est dit abstrait en ce sens que les éléments qui le constituent n'ont aucune signification intrinsèque.

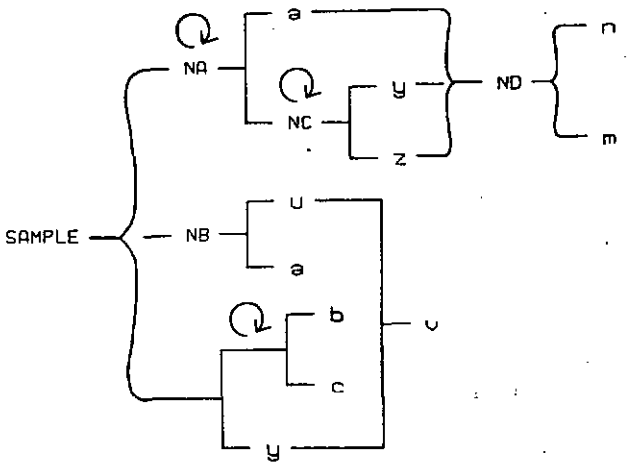


Figure 5.1

Remarques:

1) L'ensemble des noeuds SAMPLE, NA, a, NC, etc... qui figurent sur le réseau sont appelés des éléments.

2) Les éléments a et NC ne sont accessibles que si et seulement si on a "passé" au préalable par l'élément NA. Tout élément qui joue un rôle analogue à NA est appelé une condition d'entrée. Dans la figure 5.1, toutes les conditions d'entrée ont été écrites en majuscules.

3) Tout élément qui n'est pas condition d'entrée est appelé un terme.

4) Tout réseau a une condition d'entrée principale qui donne son nom au réseau. Dans le cas de la figure 5.1 il s'agit de l'élément SAMPLE. On parlera donc du réseau SAMPLE.

La première chose que BARBARA doit connaître est la condition d'entrée principale. Dans le cas de la figure 5.1, on répondrait:

START SYMBOL OF NETWORK
> sample

5) Certains éléments comme a et y apparaissent plusieurs fois dans le réseau. Ceci est licite, pour autant que l'on respecte toujours la règle suivante:

"plusieurs éléments portant le même nom ne peuvent jamais apparaître sous la même condition d'entrée".

Pour désigner un élément qui figure plusieurs fois dans le réseau, on le fait précéder de sa condition d'entrée.

On utilise le terme de noeud lorsqu'on veut désigner l'endroit où se situe un élément. Ainsi peut-on parler des noeuds NA(a), NB(a), SAMPLE(y), n, b, etc...

A noter que le noeud se confond avec l'élément lorsqu'il n'apparaît qu'une seule fois dans le réseau.

6) Considérons les "morceaux" de réseau (a, y, z, ND) d'une part et (u, y, v) d'autre part. De telles constructions s'appellent des conditions.
Une condition est toujours liée à des noeuds et non à des éléments. Par exemple, la première condition est liée aux trois noeuds: NA(a), NC(y) et z.

5.3 Vocabulaire et définitions

Vocabulaire.

Sauf s'ils apparaissent dans des commentaires, les seuls caractères admis sont: les lettres, les chiffres, les parenthèses gauche et droite, les signes égal et moins, les symboles de ponctuation point-virgule, virgule et point d'interrogation, les caractères "souligné", "barre" (slash), "blanc" et "astérisque".

Le vocabulaire de BARBARA consiste en l'ensemble des symboles suivants:

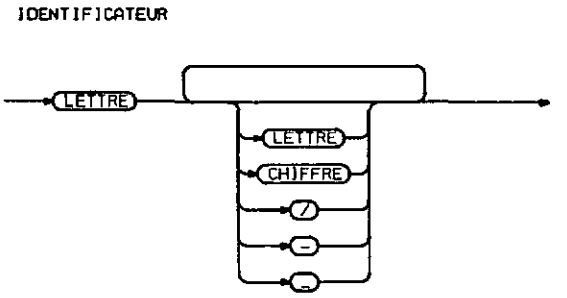
(	and	not
)	any	num
=	bar	or
,	bra	rec
?	con	tab
(*	end	ter
*)	list	

and, any, bar, etc... sont appelés des mots réservés, en ce sens qu'ils ne peuvent pas être utilisés comme identificateurs.

COMMENTAIRE.

Un commentaire est une séquence de caractères quelconques commençant par (*, se terminant par *) et ne comprenant ni le symbole ";" ni le symbole "*)". Un commentaire peut être inséré n'importe où entre deux identificateurs ou symboles quelconques. Du point de vue de BARBARA, deux textes sont syntaxiquement identiques s'ils ne diffèrent que par leurs commentaires.

Identificateur.



Un identificateur est un nom permettant de désigner:

- une condition d'entrée, généralement appelée non-terminal.

- un terme, généralement appelé terminal.

- une étiquette, unité syntaxique qui permet de repérer un ensemble de choix possibles dans un réseau.

Un identificateur doit commencer par une lettre et peut être suivi d'une combinaison quelconque de lettres, de chiffres ou de l'un des trois caractères - _ /. Bien qu'il puisse être très long, seul ses dix premiers caractères sont significatifs. Dès lors, deux identificateurs devant désigner des objets distincts ne peuvent comporter dix premiers caractères identiques.

Exemple d'identificateurs licites:

question4	pas_de_reponse	examen_du_1/12/78
pick-up	questionnaire/114	questionnaires/118

les deux derniers désignant le même objet.

Exemples d'identificateurs illicites:

3-eme_question
list

Séparateur.

Les blancs, les fins de lignes et les commentaires sont considérés comme séparateurs. Un nombre quelconque de séparateurs peut apparaître entre deux symboles consécutifs, avec l'exception suivante: aucun séparateur ne peut apparaître à l'intérieur d'un identificateur, d'un mot réservé ou d'un symbole composé comme "(*".

Exemple:

iden(* ceci est commentaire *)tificateur

sera considéré par BARBARA comme deux identificateurs: le premier "iden" et le second "tificateur".

Instruction.

Le terme général d'instruction désigne l'un des trois types suivants de données:

- une donnée visant à définir le réseau, que l'on conviendra d'appeler une règle.

- une donnée visant à définir le code, que l'on conviendra d'appeler une réalisation.

- une donnée permettant d'interroger l'ensemble réseau-code et que l'on conviendra d'appeler une question.

Une instruction peut s'étendre sur un nombre quelconque de lignes et se termine toujours par le symbole ";". Comme on le verra par la suite, une instruction oblige souvent à ouvrir un très grand nombre de parenthèses et par conséquent, théorique-

ment, à en refermer un nombre équivalent. L'équilibre entre parenthèses gauches et droites est souvent une source d'erreur. Par commodité, il a donc été décidé qu'il ne serait pas nécessaire de refermer les parenthèses droites ayant un " ". BARBARA interprète donc le symbole ";" comme:

1) fin d'instruction

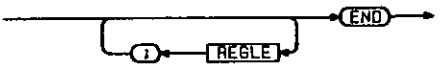
2) introduire autant de ")" qu'il reste de "(" ouvertes.

Questions posées par BARBARA.

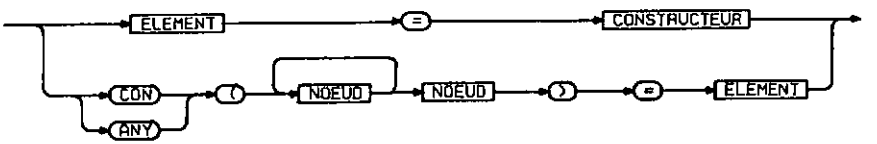
Dans un certain nombre de circonstances, BARBARA pose des questions à l'utilisateur. Par exemple, pour savoir ce qu'il doit entreprendre, pour se faire confirmer certaines situations, etc.... Aux questions qu'il pose, on peut toujours répondre par "oui" ou "non", c'est-à-dire en introduisant Y (Yes) ou N (No).

5.4 Comment introduire un réseau

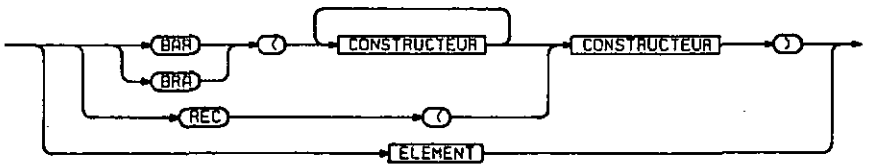
Syntaxe.



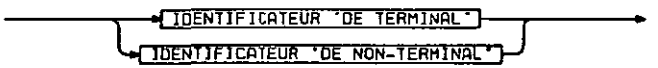
REGLE



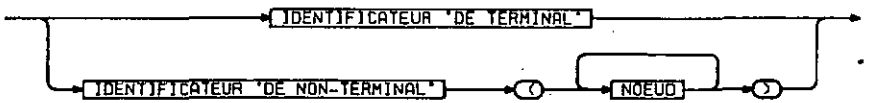
CONSTRUCTEUR



ELEMENT



NOEUD



Signification.

Tout élément qui apparaît pour la première fois dans le système y est introduit comme non-terminal s'il se situe à gauche du signe = d'une règle, comme terminal sinon.

Par contre, deux cas sont à envisager si l'on introduit un élément figurant déjà dans le système:

- il y figurait comme terminal.
  S'il apparaît à gauche du signe = d'une règle alors il passe à l'état de non-terminal, sinon il reste à l'état de terminal.

- il y figurait comme non-terminal.
  S'il apparaît à droite du signe = d'une règle, il reste à l'état de non-terminal, sinon c'est que l'utilisateur désire le redéfinir. Dans ce cas, le programme se fait confirmer qu'il ne s'agit pas d'une erreur, en écrivant le message:

```
X ALREADY DEFINED
DO YOU REALLY WANT TO CHANGE IT (Y/N)
```

Si l'on répond "oui" (Y), BARBARA détruit l'ancienne définition, pour introduire la nouvelle. Par contre, une réponse négative (N) revient à annuler l'instruction que l'on voulait introduire.

A noter que le passage de non-terminal à terminal est impossible.

Un noeud n ne peut jamais être lié qu'à une seule condition. Supposons que n ait déjà lié à une condition et que l'on tente d'introduire une nouvelle condition, dans laquelle n est impliqué. BARBARA réagit de la manière suivante:

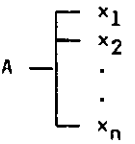
```
n ALREADY BELONGS TO
... (suit la condition à laquelle n appartient)
DO YOU REALLY WANT TO DELETE THIS CONDITION (Y/N)
```

En répondant "oui" (Y) l'ancienne condition est détruite et la nouvelle introduite; en répondant "non" (N) l'instruction courante est annulée.

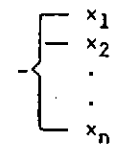
Instructions de base.

1) A = x1; traduit A — x_1

2) A = bar(x1 x2 ... xn); traduit



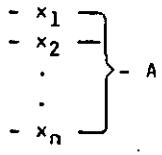
3) A = bra(x1 x2 ... xn); traduit



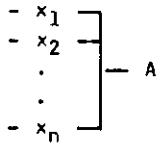
4) A = rec(x1); traduit $A \overset{\circlearrowright}{—} x_1$

Aucune hypothèse n'est faite sur les x_i. Il peut s'agir d'un élément ou d'un système, auquel cas on applique récursivement l'une des "traductions" 1 à 4.

5) con(x1 x2 ... xn) = A; traduit



6) any(x1 x2 ... xn) = A; traduit



Dans une condition, comme on le remarque sur les diagrammes syntaxiques, A doit nécessairement être un élément (terminal ou non), ceci afin de pouvoir "nommer" la condition (cf. paragraphe 5.8).

Exemple.

Après y avoir été invité par BARBARA, l'utilisateur introduirait le réseau de la figure 5.1 comme suit:

```
CREATE OR MODIFY THE NETWORK
> sample = bra(na nb bar(rec(bar(b c)) y));
> na = rec(bar(a nc));
> nc = rec(bar(y z;
```

les parenthèses droites, ici, ne sont pas refermées avant le ";"

```
> nb = bar(n a);
> any(n y) = v;
          ^
NOT SINGLE
```

en effet, "y" apparaît déjà dans la première règle. Toute erreur est signalée de manière analogue, c'est-à-dire, à l'aide d'un message en clair (en anglais) et par un pointeur qui désigne le symbole où l'erreur est apparue.

```
> any(n sample(y)) = v;
```

on s'aperçoit qu'on a confondu les lettres "n" et "u". On va donc redéfinir le non-terminal "nb".

```
> nb = bar(u a);
NB ALREADY DEFINED
DO YOU REALLY WANT TO CHANGE IT (Y/N)
> y
ANY(N Y) = V HAS BEEN DELETED
```

la redéfinition d'un non-terminal peut nécessiter la destruction de certaines conditions. Toutes les conditions détruites sont évidemment signalées par BARBARA.

```
> any(u sample(y)) = v;
> con(na(a nc(y z))) = nd;
```

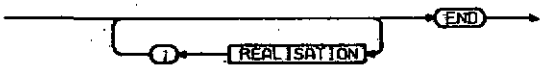
on aurait aussi pu donner con(na(a) nc(y) z) ou encore con(na(a y z)).

```
> nd = bra(n m);
> end
```

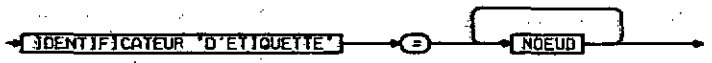
marque la fin de l'introduction du réseau.

5.5 Comment introduire le code

Syntaxe.



REALISATION



Signification.

L'ensemble des noeuds doit constituer une suite de choix possibles dans le réseau.

Les identificateurs d'étiquettes sont obligatoirement différents de ceux des éléments. La raison de cette restriction, que l'on verra au paragraphe 5.8, n'est pas de nature syntaxique. En effet, un identificateur d'étiquette ne peut jamais être confondu avec un identificateur d'élément puisqu'ils apparaissent dans des contextes tout différents.

Exemple.

Après y avoir été invité par BARBARA, l'utilisateur pourrait introduire l'ensemble des réalisations suivantes:

```
CREATE OR MODIFY THE CODE
> reponse1 = sample(ns(z)
                    nb(u) (*alors*) v
                    b c);
```

du point de vue du programme, on aurait très bien pu écrire:

```
  reponse1 = z u v b c;
```

puisque chacun des termes n'apparaît qu'une seule fois dans le réseau et qu'il ne peut, par conséquent, y avoir d'ambiguïté. Cependant, pour des raisons de clarté et de lisibilité, il est fortement conseillé de faire précéder les termes de leurs conditions d'entrée. De même, s'efforce-t-on de souligner la structure du réseau en écrivant les réalisations sur plusieurs lignes.

```
> reponse2 = sample(na(a)
                    nb(a);
```

```
EXPECTED B C Y
```

en effet, telle qu'elle a été introduite, "reponse2" est incomplète. D'une manière générale, en cas d'erreur, BARBARA donne la liste des éléments qu'il s'attend à rencontrer.

```
> reponse2 = sample(na(a)
                    nb(a)
                    y (*donc*) v;
```

la parenthèse droite n'a pas été refermée avant le ";"

```
> reponse3 = sample(nd(n m)
                    nb(a)
                    c);
```

normalement, on aurait dû introduire:

```
  reponse3 = sample(na(a) nc(y z) (*d'où*) nd(n m)
                    nb(a)
                    c);
```

cependant, le fait de "passer" par nd implique que l'on a nécessairement "passé" par les noeuds na(a), nc(y) et z. Dans un tel cas, BARBARA se charge de générer automatiquement l'ensemble des noeuds qui constituent la condition d'entrée simultanée d'un tel système.

Plus généralement, on a:

si con(n1 n2 ... nk) = c
alors c est équivalent à n1 n2 ... nk c

Remarques:

1) BARBARA génère toujours les noeuds strictement dans l'ordre: n1 puis n2 puis ... puis nk.
En particulier, si l'on considère le réseau de la figure 5.2, il n'est pas possible d'écrire x = e c, car le programme génère pour e: b suivi de d. Donc x voudrait b d c, ce qui, évidemment, est impossible dans ce réseau.

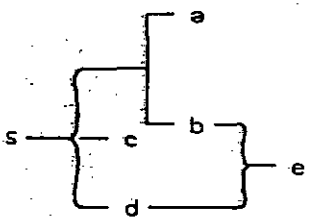


Figure 5.2

2) Cette abréviation syntaxique n'a évidemment aucun sens pour "any".

```
> reponse1 = sample(na(s)
                    nb(s)
                    b c);
REPONSE1 ALREADY DEFINED
DO YOU REALLY WANT TO CHANGE IT (Y/N)
> n
```

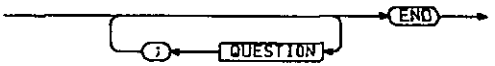
en effet, "reponse1" est une étiquette déjà utilisée dans le système. On suppose qu'il s'agit d'une erreur et non d'une redéfinition.

```
> reponse4 = sample(na(s)
                    nb(s)
                    b c);
> end
```

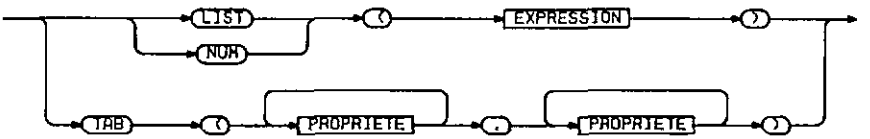
marque la fin de l'introduction du code.

5.6 Comment interroger l'ensemble réseau-code

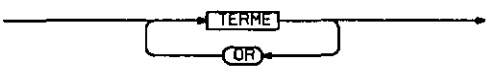
Syntaxe.



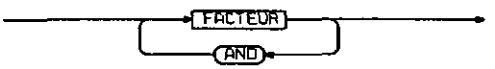
QUESTION



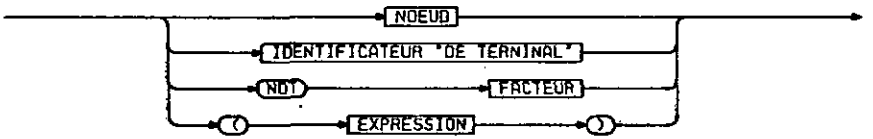
EXPRESSION



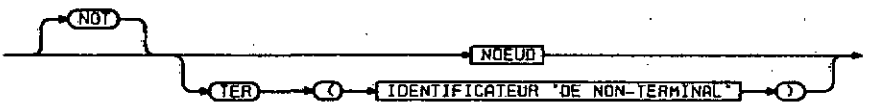
TERME



FACTEUR



PROPRIETE



Signification.

Une EXPRESSION est une expression de nature ensembliste. Elle peut être formée à l'aide des opérateurs binaires or et and et de l'opérateur unaire not. On dispose également des parenthèses gauche et droite pour mettre en évidence une sous-expression.

Soit R un réseau, A, B et C trois éléments quelconques de R. Alors on a:

- A or B est équivalent à A u B. C'est-à-dire, désigne l'ensemble des réalisations qui ont soit la propriété A soit la propiété B.

- A and B est équivalent à A n B. C'est-à-dire, désigne l'ensemble des réalisations qui ont la propriété A et en même temps la propriété B.

- not A est équivalent à R - A. C'est-à-dire, désigne l'ensemble des réalisations qui n'ont pas la propriété A.

En particulier:

- not R désigne toujours l'ensemble vide: il n'y a aucune réalisation qui n'a pas la propriété R.

- l'ensemble des réalisations qui ont la propriété A mais qui n'ont pas la propriété B s'exprime comme suit:

A and not B

On notera que les diagrammes syntaxiques sont tels qu'il y a priorité de l'opérateur and sur l'opérateur or.

Donc, A or B and C est évalué: A or (B and C).

LIST donne une liste de l'ensemble des étiquettes qui satisfont l'expression.

NUM donne le nombre d'étiquettes qui satisfont l'expression. Donc:

NUM(<expression>) = cardinal(LIST(<expression>)).

TAB permet de réaliser un tableau croisé. Un tel tableau comprend deux ensembles de propriétés, séparés par une virgule. Le premier ensemble constitue les colonnes du tableau, le second ses lignes. Le nombre des éléments en colonne est évidemment limité par la taille de l'écran du terminal à partir duquel on travaille. Ce nombre varie en fonction de la longueur de chacun des identificateurs. Par contre, le nombre des éléments en lignes est illimité.

Une propriété peut être affirmée ou niée à l'aide du mot réservé not.

Supposons que l'on désire réaliser un tableau croisé constitué de n colonnes et de m lignes. Alors on obtiendrait un résultat analogue, mais évidemment beaucoup moins synthétique, en calculant: pour tout i=1,2, ... ,n NUM(c_i and l_j) et ceci pour tout

j=1,2, ... ,m, où c_i désigne la i-ème colonne et l_j la j-ème ligne.

TER' est une fonction à un argument, un non-terminal, disons X. Elle calcule l'ensemble de tous les terminaux directement accessibles à partir de X. Si X est une condition d'entrée (non-terminal) permettant d'"atteindre" les termes (terminaux) $x_1, x_2, \ldots, x_n$ alors on a que:

- TER(X) est une abréviation syntaxique pour:
  $x_1 \quad x_2 \quad \cdots \quad x_n$

- not TER(X) est une abréviation syntaxique pour:
  not x_1 not x_2 ... not x_n

Dans le réseau de la figure 5.1 on a:

TER(SAMPLE) = b c y
TER(NA) = a
TER(NB) = u a
etc...

Exemple.

Après y avoir été invité par BARBARA, l'utilisateur pourrait poser l'ensemble des questions suivantes:

```
YOU CAN QUESTION THE DATA
> list(sample);
REPONSE1 REPONSE2 REPONSE3 REPONSE4
4 LABEL(S)
> num(b or (na(a and y;
3
```

les parenthèses droites ont été omises avant le ";"

```
> tab(ter(nb) not b, nc(y z) not tar(nd));
```

```
        U    A    NOT B
    Y   0    1      1
    Z   1    1      1
NOT N   1    2      1
NOT M   1    2      1
```

cet exemple n'a guère de sens, il n'a été donné qu'à titre purement illustratif.

```
> end
```

marque la fin de l'interrogation de l'ensemble réseau-code et par conséquent la fin du programme BARBARA.

5.7 Comment conserver les données

BARBARA offre la possibilité de conserver de façon permanente l'ensemble des données qui lui est fourni. Ceci se fait par l'intermédiaire d'un fichier de nom formel "data". En fait, on peut considérer que "data" se constitue de deux "sous-fichiers" dans lesquels les données sont conservées exactement telles que l'utilisateur les a introduites et qui, par conséquent, peuvent être imprimées et relues. On a:

- le "sous-fichier" contenant le réseau: network file

- le "sous-fichier" contenant le code: code file.

Deux "grands" cas peuvent se présenter:

i) L'utilisateur désire créer un réseau et le code lui correspondant, en rendant permanents l'un et l'autre. Il doit donc substituer (assigner) au fichier formel "data" un nom de fichier effectif, dans lequel il trouvera, à la fin de l'exécution du programme, l'ensemble de toutes les données correctes qu'il a introduit. Il est recommandé de donner comme nom de fichier effectif le nom de la condition d'entrée principale du réseau qu'il contient. Dans le cas de l'exemple

traité dans ce chapitre, le fichier effectif s'appelerait "sample".

Le programme réagit comme suit:

```
YOU HAVE AN EMPTY NETWORK FILE
IS NETWORK TO BE PERMANENT (Y/N)
> y
CREATE OR MODIFY THE NETWORK
> cf. paragraphe 5.4
```

En répondant "non", l'effet est que l'ensemble des instructions visant à créer le réseau ne devient pas permanent, c'est-à-dire disparaît à la fin de l'exécution du programme. Dans ce cas, l'utilisateur peut, s'il le veut, ne substituer aucun paramètre effectif au fichier "data". Cependant, PASCAL ne permettant pas de savoir s'il y a eu substitution ou non, BARBARA pose toujours la question (DO YOU WANT TO CREATE A NEW ONE).

Le cas du code est traité de manière analogue:

```
YOU HAVE AN EMPTY CODE FILE
IS CODE TO BE PERMANENT (Y/N)
> y
CREATE OR MODIFY THE CODE
> cf. paragraphe 5.5
```

2) L'utilisateur a déjà un fichier, contenant un réseau, qu'il désire soumettre à BARBARA. Supposons que ce fichier s'appelle "ancien". Après avoir substitué (assigné) "ancien" à "data", l'utilisateur a la possibilité de modifier ou non le réseau et de rendre ces modifications permanentes ou non.

Le programme réagit comme suit:

```
YOU HAVE A NON-EMPTY NETWORK FILE
DO YOU WANT TO CHANGE THIS FILE PERMANENTLY (Y/N)
> y
```

"oui" signifie que l'utilisateur a non seulement l'intention

de modifier le réseau, mais également l'intention de rendre ces modifications permanentes. Dans ce cas, le programme écrit:
CREATE OR MODIFY THE NETWORK
> cf. paragraphe 5.4.

> n
"non" signifie que les éventuelles modifications apportées au réseau ne seront pas rendues permanentes. Dans ce cas, le programme demande s'il doit effectuer ou non des modifications (non-permanentes) dans le réseau:
DO YOU WANT TO CHANGE THE NETWORK (Y/N)
> y
si "oui", alors le programme écrit:
CREATE OR MODIFY THE NETWORK
> cf. paragraphe 5.4.
> n
si "non", alors le programme passe directement au traitement du code.

Le fait que "ancien" contienne un réseau n'implique pas nécessairement qu'il contienne le code qui lui correspond. Donc, deux cas sont à envisager:

a) le fichier ne comporte aucune réalisation. On est donc ramené au cas traité sous 1 (création d'un fichier-code).

b) le fichier comporte déjà un certain nombre de réalisations. Dans ce cas, l'algorithme est identique à celui que l'on vient de voir sous 2, avec les deux différences suivantes:

- le mot NETWORK est systématiquement remplacé par le mot CODE.

- les références au paragraphe 5.4 deviennent toutes des références au paragraphe 5.5.

Le programme BARBARA se déroule en trois phases:

a) introduction du réseau
b) introduction du code
c) interrogation de l'ensemble réseau-code

Le passage d'une phase à l'autre se fait automatiquement et a toujours lieu dans le sens a - b - c. Il est bien évident que si aucun réseau n'a été introduit, le programme ne passe pas à la phase b. Il imprime le message:

NO DEFINED NETWORK

Respectivement, si aucune réalisation n'a été introduite, il ne passe pas à la phase c et imprime le message:

NO DEFINED CODE

Il n'est pas possible de "sauter" une des phases ou de "revenir en arrière". Par contre, il est toujours possible de sortir immédiatement d'une phase en introduisant le mot réservé end et de relancer l'exécution du programme en lui donnant comme fichier celui que l'on était en train de créer, pour autant que ce fichier existe. Cette manière de faire offre certains avantages:

- la définition d'un réseau ou du code peut se faire en plusieurs étapes, donc à des moments différents.

- cela permet de "revenir en arrière" dans les phases. Par exemple, de modifier le réseau alors qu'on est en train de définir le code.

- lorsque les données à introduire représentent un volume assez considérable, il est toujours prudent de prévenir un éventuel "crash" (système d'exploitation ou programme lui-même). Il est donc fortement conseillé d'interrompre parfois l'exécution, de manière à sauver (sur disque) le fichier qui contient l'ensemble des données déjà introduites.

Définitions:

1) une règle qui n'est sccessible depuis sucun chemin psrtsnt de la condition d'entrée principale eat dite: une règle non connexe.

2) supposons qu'un fichier contienne non seulement un réseau mais également un certain nombre de réslisations. Supposons, par ailleurs, que l'on modifie le réseau. Alors, les réslisstions que comporte le fichier peuvent ne plus correspondre au réseau modifié. Dsns ce cas, on parle de réalisations incorrectes ou, psr sbus de lsngsge, d'étiquettes incorrectes.

A ls fin de chaqus phass, BARBARA donne un certain nombre de renseignements:

a) introduction du réseau.
Le progrsmme signsle s'il y s des règles non connexes ou pss:

soit: ALL THE RULES CONNECTED

soit: n NON-CONNECTED RULE(S)
et si le réseau eat créé de façon permanente:
DO YOU WANT TO KEEP THE NON-CONNECTED
RULE(S) IN THE NEW FILE (Y/N)
> y

le fait de répondre "oui" signifie que l'on a l'intention de rendre les règles connexes lors d'une modification ultérieure du réseau.

b) introduction du code.
le programme signale s'il y a des réalisations incorrectes ou non:

soit: ALL THE ITEMS MATCH THE NETWORK

soit: n ITEM(S) DO NOT MATCH THE NETWORK
et si le code aat créé de fsçon permanente:

```
DO YOU WANT TO KEEP THE NON-MATCHING
ITEM(S) IN THE NEW FILE (Y/N)
>
```

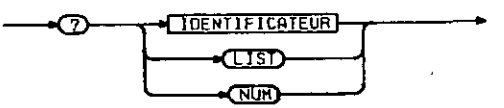
c) interrogation de l'ensemble réseau-code, c'est-à-dire: fin du programme.
BARBARA signale si un nouveau fichier a été créé ou effectivement modifié:

soit: YOUR FILE IS EMPTY.
Ce serait le cas, par exemple, d'un réseau permanent qui ne comprendrait que des règles non connexes dont on aurait demandé l'élimination.

soit: DON'T FORGET TO SAVE THE FILE CREATED.
Sur certaines installations, il convient de prendre des mesures spéciales pour s'assurer que le fichier ne soit jamais détruit.

5.8 Comment questionnner BARBARA

Syntaxe.



Signification.

Chaque fois que BARBARA pose une question ou après avoir introduit une instruction quelconque, il est possible d'obtenir des renseignements:

- sur les objets (éléments ou réalisations) que manipule le programme. C'est la raison pour laquelle les identificateurs d'étiquettes doivent être tous différents de ceux des éléments (cf. paragraphe 5.5).

- sur la définition courante donnée à chacun de ces objets et, s'il s'agit d'éléments, le nombre de fois que l'on s'y réfère.

- sur l'ensemble courant des règles non connexes et si cet ensemble est non vide, sur les règles qu'il contient.

- sur l'ensemble courant des réalisations incorrectes et si cet ensemble est non vide, sur les réalisations qu'il contient.

Lorsqu'on entre un point d'interrogation suivi d'un identificateur, disons X, BARBARA réagit comme suit:

- X est inconnu:

  UNKNOWN

- X est connu:

  1) il s'agit d'un identificateur d'élément:

     a) il s'agit d'un terminal
        X TERMINAL
        n REFERENCE(S)
        ce qui signifie que X apparaît n fois dans le réseau.

     b) il s'agit d'un non-terminal
        X= ... (suit l'expression que contient X)
        n REFERENCE(S)
        n est le nombre de fois où X apparaît dans le réseau, y

comprls sa définition. Ainsi, dans l'exemple traité dans ce chapitre, chaque non-terminal n'apparaît qu'une seule fois dans le réseau, donc pour tout non-terminal, n vaut un. A noter que:

- s'il s'agit de la condition d'entrée principale, le message ci-dessus est précédé de:
  START SYMBOL
  et dans ce cas seulement, n est égal à zéro.

- l'expression que contient un non-terminal est toujours simplifiée. Ainsi, si l'on a introduit X = bar(bar(x1 x2)) alors X est mémorisé et donc récrit X = bar(x1 x2).

c) il s'agit d'un élément (terminal ou non) que l'on atteint par une condition, c'est-à-dire que l'on a: soit con(n1 n2 ... nk) = X soit any(n1 n2 ... nk) = X. Dans ce cas, BARBARA écrit d'abord la condition puis X est traité comme en 1a), ou en 1b), selon qu'il s'agit d'un terminal ou non.

2) il s'agit d'un identificateur d'étiquette:

le programme récrit l'ensemble des options choisies dans le réseau. La réalisation n'est pas donnée telle que l'utilisateur l'a introduite (mise en pages, conditions d'entrée précédant les termes, passage à la ligne, etc...), elle est toujours simplifiée à l'extrême: les noeuds ne sont jamais précédés de leur condition d'entrée sauf s'il y a ambiguïté. Cette remarque s'applique également aux noeuds en paramètres d'une condition.

?LIST fournit les renseignements suivants:

- si l'on est en train de définir le réseau:
  Donne la liste des règles non connexes. Donc:
  - soit ALL THE RULES CONNECTED
  - soit REGLE1 REGLE2 ... REGLEN
    N NON-CONNECTED RULE(S)

où les REGLEi sont des identificateurs de non-terminaux

- si l'on est en train de définir le code:
  Donne la liste des réalisations incorrectes. Donc:
  - soit ALL THE ITEMS MATCH THE NETWORK
  - soit REAL1 REAL2 ... REALN
    N ITEM(S) DO NOT MATCH THE NETWORK
    où les REALi sont des identificateurs d'étiquettes

- si l'on est en train d'interroger l'ensemble réseau-code:
  BARBARA imprime toujours les deux messages:
  ALL THE RULES CONNECTED
  ALL THE ITEMS MATCH THE NETWORK
  En effet, les éventuelles règles non connexes, respectivement les éventuelles réalisstions incorrectes sont toujours supprimées (en mémoire) avant de pssser à la dernière phase du programme.

?NUM réalise exactement la même fonction que ?LIST si ce n'est que le programme ne donne pas la liste des identificateurs, des non-terminaux, respectivement des étiquettes.

Exemple: on suppose que l'on est en train d'interroger l'ensemble réseau-code.

```
> ?sample
  START SYMBOL
  SAMPLE=BRA(NA NB BAR(REC(BAR(B C)) Y));
  0 REFERENCE(S)
> ?nb
  CON(NA(A) NC(Y) Z)=NO;
  NO=BRA(N M);
  1 REFERENCE(S)
> ?a
  A TERMINAL
  2 REFERENCE(S)
> ?truc
  UNKNOWN
> ?reponse1
  REPONSE1=Z U V B C;
```

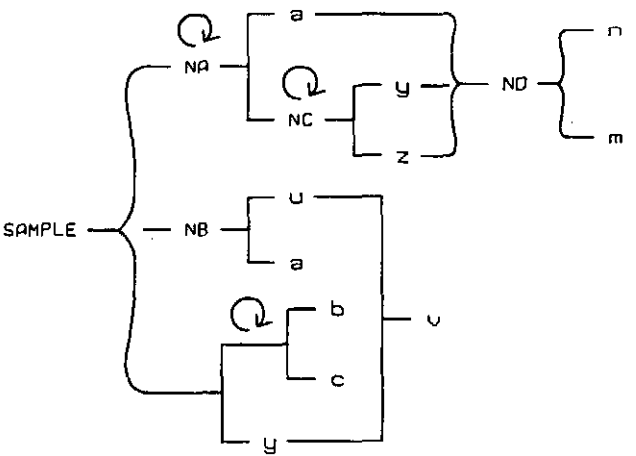
```
> ?reponse2
  REPONSE2=NA(A) NB(A) SAMPLE(Y) V;
> ?reponae3
  REPONSE3=NA(A) NC(Y) Z N M NB(A) C;
```

En complément de ce chapitre, on trouvera dans l'annexe I des exemples commentés de dialogues entre le programme et l'utilisateur.

ANNEXES

Cette annexe comporte deux exemples de dialogue avec BARBARA. Le texte en minuscules est introduit par l'utilisateur, celui en majuscules est écrit par le programme. On trouve à droite un certain nombre de commentaires.

Le réseau qui sert d'exemple est repris de la figure 5.1. Le premier dialogue contient un grand nombre d'erreurs contrairement au second qui n'en comporte aucune.



Dialogue avec erreurs

```
PROGRAM BARBARA V2.4 08/06/80                version 2, modification 4

START SYMBOL OF NETWORK                      condition d'entrée principale du réseau
```

```
>aamplet
THE START SYMBOL IS SAMPLET PLEASE CONFIRM (Y/N)
>n
START SYMBOL OF NETWORK
>aample
THE START SYMBOL IS SAMPLE PLEASE CONFIRM (Y/N)
>y
YOU MAVE AN EMPTY NETWORK FILE
IS NETWORK TO BE PERMANENT (Y/N)
>yea
 ^
ANSWER BY "Y" OR "N"
>y
CREATE OR MODIFY THE NETWORK
>aample:=bra(rec(na) nb bar(rec(bar(b c)) y));
       ^
= EXPECTED
>aample=bra(rec(na) nb bar(rec(bar(b c)) y));
>na=baar(a rec(nc));
        ^
; EXPECTED
>na=bar(a rec(nc;
>nb=bar(u);
        ^
CON,ANY,BAR OR BRA MUST MAVE AT LEAST 2 ARGUMENTS
>nb=bar(u a);
>?tar
  ^
A WORO, LIST OR NUM EXPECTEO
>?xyz
UNKNOWN
```

demande confirmation en cas d'erreur de frappe
on a'est trompé

cette foia, la condition d'entrée principala
a été introduita correctement

on désire créer un réaeau et le conaervar aur fichier

ERREUR: le programme ne comprend que Y ou N

INTROOUCTION OU RESEAU

ERREUR ayntaxique

ERREUR: le programme considère cetta inatruction comme:
na = baar; d'où le message
il est poaaible de ne pas fermer les) avant un ;

ERREUR ayntaxique

ERREUR ayntaxique: TER eat un mot réaervé

inconnu dans le réaeau

```
>?nc                                          pour savoir si nc a déjà été introduit
NC TERMINAL                                  non puisqu'il est à l'état de terminal
1 REFERENCE(S)                               n'est utilisé qu'une seule fois dans le réseau
>con(a y z)=nd;
     ^
NOT SINGLE                                   ERREUR: a figure deux fois dans le réseau
>con(na(a) nc(y z))=nd;                      donc le programme ne peut savoir duquel il s'agit
            ^
NOT CORRECT AFTER A TERMINAL                 ERREUR: nc est un terminal, il ne peut donc être
>nc=bar(yl z);                               utilisé comme condition d'entrée
>con(na(a) nc(yi z))=nd;                     on définit nc
>nc=bar(y z);                               puis on donne la condition
NC ALREADY DEFINED                           on redéfinit nc, car on a introduit yl à la place de y
DO YOU REALLY WANT TO CHANGE IT (Y/N)        le programme constate que nc a déjà été défini
>y                                           il demande donc s'il doit supprimer l'ancienne définition
CON(NA(A) Yl NC(Z))=ND HAS BEEN DELETED      en l'occurrence oui
>con(na(a) nc(y z))=nd;                     la redéfinition de nc détruit la condition
>nd=bra(n n);                               qu'il faut donc réintroduire
          ^
APPEARS TWICE UNDER THIS ENTRY               ERREUR: sous une même condition d'entrée
>nd=bra(n m);                                chaque terme doit être différent
>any(toto y)=v;
     ^
UNKNOWN WORD                                 ERREUR: tous les termes qui figurent
>any(nb(u y))=v;                             dans une condition doivent être connus
          ^
INACCESSIBLE FROM NB                         ERREUR: aucun y n'est accessible à partir de nb
>any(nb(u) sample(y))=v;                     définition de la seconde condition
>?nd                                         on s'assure que nd est défini
CON(NA(A) NC(Y) Z)=ND;                       nd repère une condition
ND=BRA(N M);                                 définition de nd
```

```
1 REFERENCE(S)
>y1=toto;;
         ^
WARNING THIS SYMBOL AND THE FOLLOWING WERE IGNORED
>end
1 NON-CONNECTED RULE(S)
DO YOU WANT TO KEEP THE NON-CONNECTED RULE(S)
IN THE NEWFILE (Y/N)
>?list
Y1
1 NON-CONNECTED RULE(S)
>n
YOU HAVE AN EMPTY CODE FILE
IS CODE TO BE PERMANENT (Y/N)
>y
CREATE OR MODIFY THE CODE
>x1=na(a);
        ^
EXPECTED NB(...)
>x1=na(a) nb(a) y;
               ^
NOT SINGLE
?y
Y TERMINAL
2 REFERENCE(S)
>x1=na(a) nb(a) sample(y);
                        ^
EXPECTED V
>x1=na(a) nb(a) sample(y) v;
>sample=z nb(a) b c;
 ^
```

nd n'est utilisé qu'une seule fois

il ne s'agit pas d'une erreur mais d'un avertissement
donc la ligne est acceptée, mais avec un ; seulement
FIN DE L'INTRODUCTION DU RESEAU
le programme signale une règle non-connexe

pour connaître le nom de cette règle

on ne la garde pas dans le fichier
on désire créer le code et le conserver sur fichier

INTRODUCTION DU CODE

ERREUR: attend le second élément du système-bra

ERREUR: y est utilisé plusieurs fois dans le réseau

pour savoir combien de fois

ERREUR: attend le système-any

ERREUR: les noms d'étiquettes doivent être

```
IDENTIFIER OF THE NETWORK
>x2=z ns(a) b c v;
                ^
NOTHING ELSE EXPECTED
>x2=z nb(a) b c;
>x3=nd(n m) nb(a) c;
>?x3
X3=NA(A) NC(Y) Z N M NB(A) C ;
>list=nc(y z) u v c;
 ^
A WORD IS EXPECTED
>x4=nc(y z) u v c;
>end
ALL THE ITEMS MATCH THE NETWORK
YOU CAN QUESTION THE DATA
>sample;
 ^
LIST, NUM OR TAB EXPECTED
>list(sample);
X1 X2 X3 X4
4 LABEL(S)
>list(b not c);
       ^
) EXPECTED
>list(b or and c);
          ^
ERROR IN FACTOR
>list(b and not c);
0 LABEL(S)
>?x1
X1=NA(A) NB(A) SAMPLE(Y) V ;
```

différents de ceux des noeuds du réseau

ERREUR: plus rien n'est attendu après c

le programme génère automatiquement la condition
pour vérifier

ERREUR syntaxique: list est un mot réservé

FIN DE L'INTRODUCTION DU CODE
l'ensemble du code introduit est correct
DEBUT INTERROGATION DE L'ENSEMBLE RESEAU-CODE

ERREUR syntaxique

donne l'ensemble de toutes les étiquettes introduites

ERREUR syntaxique: not ne peut commencer une expression
le programme pense donc qu'elle est terminée

ERREUR: expression logique incorrecte

pour connaître l'ensemble de toutes les étiquettes
qui sont b sans être c
pour savoir comment x1 était défini

```
>tab(na ter(v) , ter(sample));
                ^
MUST BE NON-TERMINAL
>?v
ANY(U Y)=V;
V TERMINAL
1 REFERENCE(S)
>tab(na ter(nd) , ter(aample));

              NA N M
            B 1  0 0
            C 3  1 1
            Y 1  0 0

>end
OON'T FORGET TO SAVE TME FILE CREATED
```

ERREUR: ter exige un non-terminal comme argument

on peut aavoir comment le réseau est défini

la fonction tab comprend deux listes de noeuds
la première liste constitue les colonnes du tableau
la seconde ses lignes

FIN OU OIALOGUE

Oialoque aans erreur

```
PROGRAM BARBARA V2.4 08/06/80

START SYMBOL OF NETWORK
>aample
TME START SYMBOL IS SAMPLE PLEASE CONFIRM (Y/N)
>y
YOU HAVE AN EMPTY NETWORK FILE
IS NETWORK TO BE PERMANENT (Y/N)
>y
CREATE OR MODIFY THE NETWORK
```

```
>aample=bra(rec(na) nb bar(rec(bar(b c)) y));
>na=bar(a nc);
>nc=bar(y z);
>con(na(a) nc(y z))=nd;
>nd=bra(n m);
>nb=bar(u a);
>any(u sample(y))=v;
>end
ALL TME RULES CONNECTED
YOU HAVE AN EMPTY CODE FILE
IS COOE TO BE PERMANENT (Y/N)
>y
CREATE OR MODIFY THE CODE
>x1=na(a) nb(a) sample(y) v;
>x2=z nb(a) b c;
>x3=nd(n m) nb(a) c;
>x4=nc(y z) u v c;
>and
ALL THE ITEMS MATCH THE NETWORK
YOU CAN QUESTION THE DATA
>liat(sample);
X1 X2 X3 X4
4 LABEL(S)
>liat(b and not c);
0 LABEL(S)
>list(not (b and c));
X1 X3 X4
3 LABEL(S)
>num(b or c);
3
>tab(na ter(nd) , ter(sample));
```

```
              NA N M
            B 1  0 0
            C 3  1 1
            Y 1  0 0

>tab(not ter(nc) not v , b c);

              NOT Y NOT Z NOT V
            B   1     0     1
            C   1     0     2

>end
DON'T FORGET TO SAVE THE FILE CREATED
```

ANNEXE II

Cette annexe illustre l'utilisstion de réseaux systémiques dans un domaine qui peut s'apparenter à celui des banques de données. Elle décrit une recherche, en coura d'élaboration, menée par le Chelsea College - Centre for Science Education - Université de Londres. Elle est tirée de:

OGBORN, J., Use of a systemic network for managing an item bank.

The problem

The Nuffield Advanced Physics Project provides a national examination at Advanced Level (age 18) for more than 8000 candidates annually. One part of the examination is a 40-item multiple choice paper. Questions (items) for this paper are produced and used as follows:

- draft items written by item writers (about 100 per year)
- draft items edited into pretest packages (3 x 30)
- items pretested and pretest data computed
- items stored, rewritten and retested, or scrapped
- examination paper constructed from tested and used items

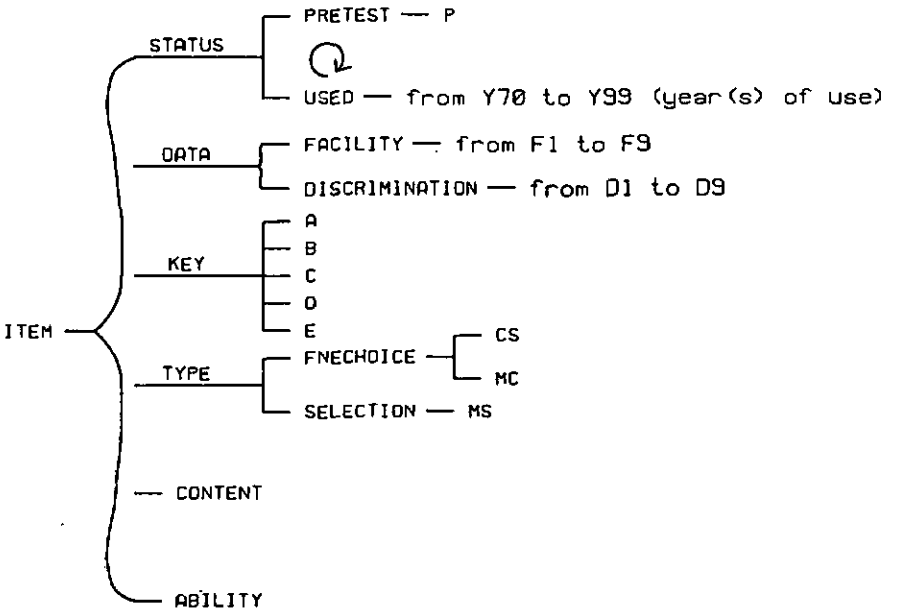
The examination has been in existence since 1970, and so has (1980) 400 items used in examinations, some 100 items pretested but not yet used, and some 100 new items coming in each year. Thus a pool of some 600 items needs to be managed, both for selecting items for new papers and for commissioning new items to fill gaps in the pool.

The conventional systems in existence, and which we use, handle the basic management data well. It is not difficult to store item serial numbers together with such data as pretest status or year(s) of use, statistics such as the item facility (percentage correct response) and discrimination (correlation with total test score of correct response), the key (correct response), and the item type (multiple choice, multiple selection or classification set).

More serious problems arise in storing the content of the item, and the abilities it calls for. The first (content) is simply a

problem of complexity: the current systems provide typically a simple alphabetic code letter, so offering a 26 single category system. It is rather easy to exhaust such possibilities, and if one increases their number then the system becomes impossible to remember. Worse, the structural relations between questions are not represented. That is, for example, such a system cannot "know" that one item about electric circuits and another about electric fields are both about electricity, which is itself related to electromagnatism. It is precisely here that systemic networks have something to offer.

The second problem, of defining and storing abilities required by items is difficult both because any encoding has a complex structure, and - much more important - because nobody has a sufficient psychology of physics questions to generate a good scheme. This problem will be discussed no further here.



Management data

The translation of the management data, excluding item content and abilities, into a systemic network is trivial.

Facilities and discriminations are more than 0 and less than 1; for the present purpose one digit precision is sufficient.

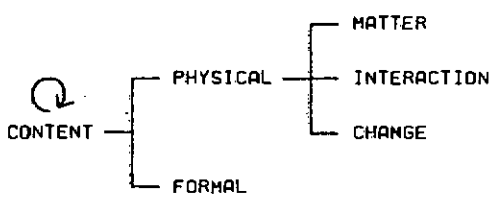
Content description

The structuring of a network to describe the content of items is necessarily some kind of theory of the subject; a form of item epistemology. It is relatively easy to construct (by contrast with a description of abilities) because physics is a subject with strong explicit order. It is, however, an epistemology for a restricted purpose, namely that of describing content differences which are meaningful for the examination paper in question. It is not a general epistemology of physics.

For Nuffield physics, the three main characteristics of content (the three main epistemological categories) are:

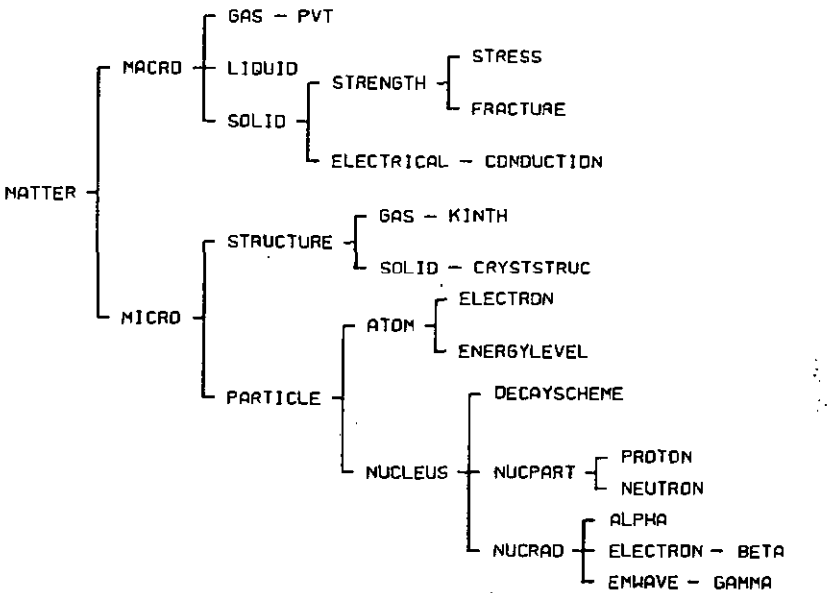
MATTER
INTERACTION
CHANGE

together with a "formal" category for mathematical and abstract manipulations. An item can involve features from more than one category, so the content network needs to be recursive.



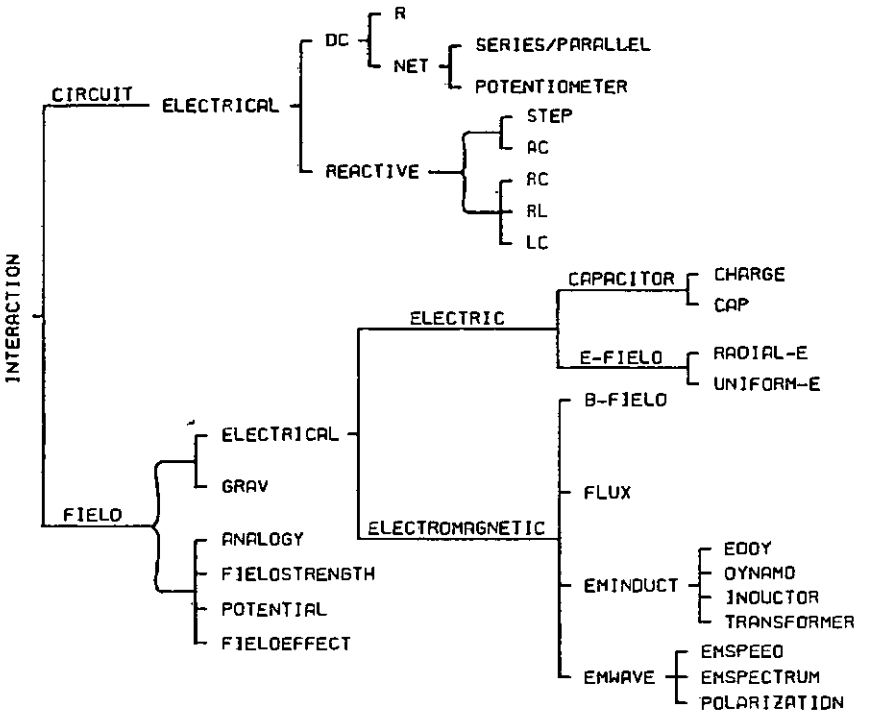
The system MATTER includes both macroscopic properties, and microscopic structure and constituents. It shares certain featu-

rea (e.g. ELECTRICAL) with the system for INTERACTION - simply because matter is composed of charged particles.

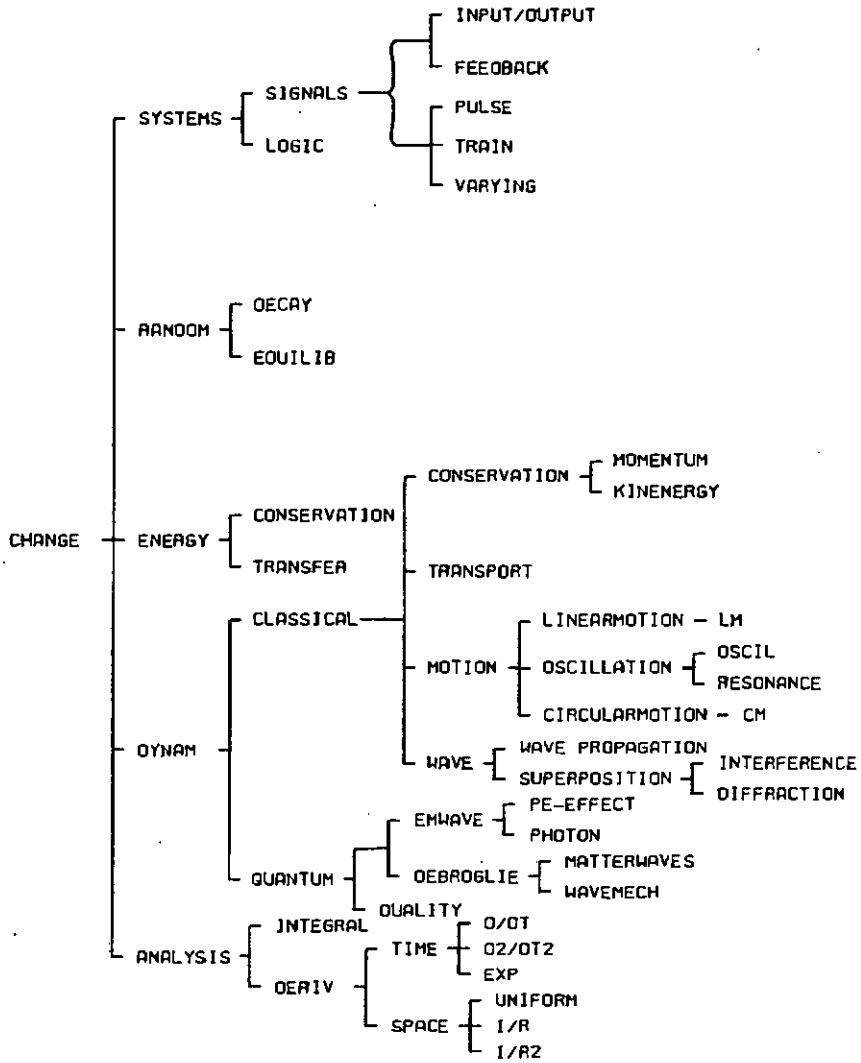


Notice the need for features in more than one context, for example MACRO(SOLID) is not the same as MICRO(STRUCTURE(SOLID)), or ATOM(ELECTRON) compared with NUCRAD(ELECTRON)-BETA.

The system INTERACTION deals with fields (electric, magnetic and gravitational, together with electric current and charge).



The system CHANGE deals with types of analysis of processes of change and conservation, together with statistical processes and processes in system structures (electronics).


CHANGE
SYSTEMS
SIGNALS
LOGIC
INPUT/OUTPUT
FEEDBACK
PULSE
TRAIN
VARYING
RANDOM
DECAY
EQUILIB
ENERGY
CONSERVATION
TRANSFER
DYNAM
CLASSICAL
CONSERVATION
MOMENTUM
KINENERGY
TRANSPORT
MOTION
LINEARMOTION - LM
OSCILLATION
OSCIL
RESONANCE
CIRCULARMOTION - CM
WAVE
WAVE PROPAGATION
SUPERPOSITION
INTERFERENCE
DIFFRACTION
QUANTUM
EMWAVE
PE-EFFECT
PHOTON
DEBROGLIE
MATTERWAVES
WAVEMECH
QUALITY
ANALYSIS
INTEGRAL
DERIV
TIME
D/DT
D2/DT2
EXP
SPACE
UNIFORM
I/R
I/R2

Coding of sample items

Codings using this network are given for some items from the 1980 Nuffield physics multiple choice paper.

```
 I1=Y80 F9 D3 KEY(A) TYPE(CS) STRESS
 I3=Y80 F8 03 KEY(C) TYPE(CS) OSCIL
 I6=Y80 F6 D4 KEY(8) TYPE(CS) ENERGYLEVEL and PHOTON
 I7=Y80 F8 D3 KEY(8) TYPE(CS) STEP RC
I10=Y80 F8 D3 KEY(C) TYPE(CS) B-FIELD and FLUX
I12=Y80 F5 83 KEY(C) TYPE(CS) GRAv POTENTIAL and I/R
I13=Y80 F6 D4 KEY(E) TYPE(MC) ENERGY(TRANSFER) and INTEGRAL
I14=Y80 F5 D4 KEY(C) TYPE(MC) FORMAL
I15=Y80 F6 D4 KEY(8) TYPE(MC) CIRCULARMOTION
                              and GRAV FIELDSTRENGTH
I16=Y80 F6 D4 KEY(C) TYPE(MC) DIFFRACTION
I21=Y80 F6 D5 KEY(D) TYPE(MC) STEP LC and RESONANCE
I22=Y80 F5 D4 KEY(E) TYPE(MC) POTENTIOMETER
I25=Y80 F9 03 KEY(A) TYPE(MC) DECAYSCHEME
I28=Y80 F8 02 KEY(E) TYPE(MC) FORMAL and WAVEPROPAGATION
I32=Y80 F8 D3 KEY(C) TYPE(MS) UNIFORM-E
I33=Y80 F8 D3 KEY(E) TYPE(MS) TRANSFORMER
I34=Y80 F7 D4 KEY(D) TYPE(MS) B-FIELD
i36=Y80 F6 04 KEY(D) TYPE(MS) EQUILI8
I40=Y80 F6 04 KEY(8) TYPE(MS) WAVEMECH
```

Remarks:

I) Note that the very tree-like nature of the network leads to brief highly specific content codes (in general one terminal per item, with a few more complex cases where recursion is used or where a "bra" is involved.
This is a deliberate choice, to keep the work of encoding simple, and to reflect the fact that physics has a very hierarchical structure. It has led to some "hidden" structure in the network, where terms are repeated so as to attribute a general property to features which are not on the same branch of the tree (see e.g. EMWAVE in both CHANGE and INTERACTION). It may reflect a defect in the thinking behind the network, or an inescapable problem.

2) updating
For the purpose of an item bank, programme facilities for updating codes become essential. For example, if an item is used twice it becomes necessary to change the Year code by adding another. Also, each year a batch of items needs to be added, and some to be deleted. It would thus be convenient to have an append function to call a file of code and add it to an existing code file. At present this has to be done using the system editor - that may be the best way in the end.

3) The main advantage of the content codes for practical use is their highly specific nature. It is easy to code items, since the code reflects very closely the item content. By further extension of delicacy the correspondence could be made even closer.

ANNEXE III

Cette annexe décrit la première partie d'une recherche menée au Chelsea College - Centre for Science Education - Université de Londres. Elle est tirée de:

MONK, M., Network analysis of the "People in my class" questionnaire data.

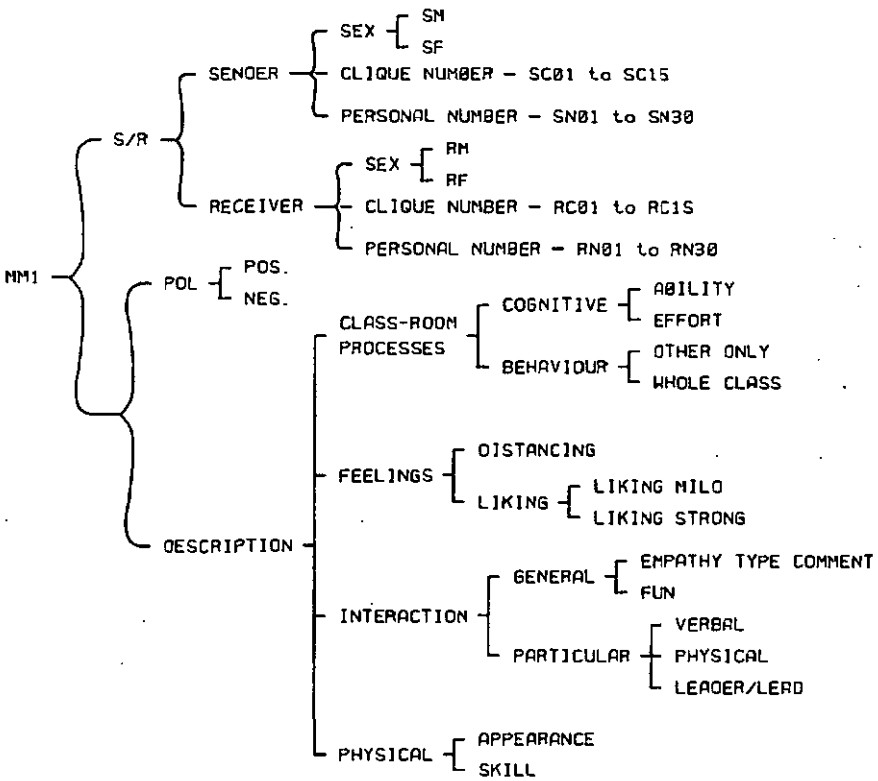
Cette recherche relève essentiellement de la psychologie; elle vise à étudier l'image que des écoliers se font d'eux-mêmes face à l'apprentissage scolaire. Les réseaux systémiques permettent de dégager de nombreuses catégories et relations entre individus.

Network analysis

The raw data consisted of as many single sheets of paper as there were children in the class, one per child. The names of all the children in the class had been listed on the sheet, arranged according to the clique and friendship structure obtained from the sociogram questionnaire. Alongside each name the child had written a free response comment on his/her peers.

The raw data presented two problems for further analysis; firstly the diversity of the free response comments had to be captured and simplified in some systematic coding process; secondly the responses needed to be sorted in a variety of ways according to the various groupings, half-class, friendship group, clique, sex.

The systemic coding process required that some network be constructed out of the data, so that it could be used to reflect the important features of the data and then quantify, in various ways, those features, as they occur in the various groupings previously mentioned. To this end the final network consists of two main parts: firstly the DESCRIPTIVE part that arises out of the variety of free responses; and secondly the sender/receiver (S/R) part that enables various groupings to be arrived at.



Taking the S/R part of the network first, it can be seen that it falls into two very similar parts, SENDER and RECEIVER. For each of these parts there are three labels acting as co-selections. Each child in the class carries a PERSONAL NUMBER, as well as a CLIQUE NUMBER, and his/her SEX identifier. When one child writes a comment about another, each of the three identifiers need to be known for both the child writing, and the child being written about. There is a certain amount of redundancy here but it does help considerably in defining new parameters for analysis. So it would be possible to get by with the childs identifying number

only, but then the simple sorting by sex, say, would require considersbly more elaborate definitions, later at the questioning stage.

Considering now the descriptive part, this is the psrt of the network thst is much more difficult to develop and requires an intimate knowledge of the data. As three clsses were being simultaneously studied, a single network of this sort silows for scross cisss compariaons. It was decided that the dats preaented four msin types of free response; those that were to do with CLASSROOM PROCESS; those that were to do with outward going FEELINGS; those that were to do with aspects of the other chiids personaiity as manifeat in socisi INTERACTION; snd lsstly those to do with PHYSICAL aspecta of the other child.

Perhaps the essisst to desl with is the physicsi sspecta. For thia there are two psrts, either comments to do with ths othera SKILLS in some phyaical way, such as swimming, sthletics, sports in gsneral or, comments to do with the others APPEARANCE, such as good looks, "handsome", "looks like a muppette", etc.

The cisasroom processes are divided into two types: those to do with COGNITIVE aspects and those to do with BEHAVIOURAL aspects. The cognitive aspecta are again divided into two; thoae to do with ABILITY, auch ss comments like "brainy", "dim" etc, and thoae to do with EFFORT, auch as "works hard", "just doesn't try".

The behavioural aspects fall into two terminal categories; those to do with the OTHER ONLY, auch as "always chewing", "chatters" snd thoae to do with the WHOLE CLASS, auch as "plays about", "very noisy".

The feelings category is made more delicate through a division into DISTANCING, and LIKING. Distancing comments are of the sort "don't have much to do with him", "admire her" etc. Whilst the liking category is subdivided into two further terminal categories of MILD liking, such as "okay", "nice" etc and STRONG liking such as "good friend", "very very nice" or negatively "hate".

The interaction category is divided into two further levels of delicacy. At the first level there is GENERAL and PARTICULAR and at the second level EMPATHY, FUN, VERBAL, PHYSICAL, LEADER/LEAD. The empathy terminal is used to code comments such as "understanding" "help you" or negatively "tell tale", "bitchy". The fun terminal is self explanatory. In the negative sense "fool", "silly" would be coded under fun. Empathy and fun together are associative types of comment. Verbal, physical, leader/lead, are more to do with domination. Verbal is used to code "big head", "show off" etc in the positive sense "nice to talk to". Physical is used to code "bully", "thinks he is tough" type comments. Lastly, leader/lead is used to code comments on leadership, or people being easily led.

For each of these terminals in the description there is a POLARITY that can be either POSITIVE or NEGATIVE.

Questioning the data

The class can be considered in the form of a matrix with the two axes being sender and receiver and the cells being filled with comments. In questioning the data it is necessary to subdivide the class into various different groupings and compare comments.

Consider the situation if the class is for some reason divided into three groups. From the data we need to know what the rest of the class thinks about each group, what that group thinks about itself and what that group thinks about the rest of the class. These three aspects are the "descriptors", "selves" and "constructs" that are arrived at from the questioning of the data.

The importance of the network in sorting the data is made clear when the groupings are changed, for then it is important that the comments are resorted into the correct new aspects of descriptors selves and constructs, with out any loss or overlap.