ISSN 0291 - 0415

Informatique
fondamentale
et
Applications

Editeur
L.I.T.L.

Depositaire

G. Ambard

Sommaire

Décembre 1982

Faculté des Sciences de Luminy
Département de Mathématique-Informatique
70, route Léon-Lachamp - Case 901
13288 MARSEILLE CEDEX 9

Editorial

e. bianco

INFORMATIQUE ET PSYCHOLOGIE.

L'être et le paraître. Aspect fondamental de l'informatique et déja source de contradictions. Les "vrais" informaticiens savent bien qu'en informatique on ne traite que le paraître des choses, et non les choses elles-même. On soumet à l'automate une "représentation des choses", et une représentation est déja le résultat d'un choix tellement arbitraire. Penser qu'il y a une réalité des choses - c'est là une attitude qui en vaut bien une autre - revient par là-même à se demander ce qu'il reste de cette réalité qui va être soumis à l'automate.

J'ai employé l'expression de vrai-informaticien. Puisque je l'emploie, je suis censé lui donner un sens. Me plaçant dans ce cas-là, je laisse sous-entendre qu'il existe des informaticiens qui ne sont pas vrais, pour la commodité du propos je dirai les faux informaticiens. Quel que puisse être mon point de vue sur la question une évidence surgit: l'expression employée est une sorte d'étiquetage d'une personne.

J'irai plus loin, l'expression décrit un personnage. Et nombre d'entre nous auront pu en endosser le costume. Mais ne s'agit-il pas, en fait, d'une représentation? On condense la personnalité de Monsieur X, qui a toutes chances au demeurant d'être fort complèxe, en disant de lui: c'est un vrai-informaticien.

Je ne vois d'ailleurs pas pourquoi, d'autres observateurs, armés d'une bonne foi également évidente, ne seraient pas amenés à proclamer: Monsieur X est un faux-informaticien.

Qu'en est-il en réalité? ce n'est point l'objet de mon propos. Ce qui m'amène

à dire ces choses provient du souvenir d'une scène à laquelle j'avais assisté il y a des temps de cela, lors d'une réunion d'informaticiens dont le but était de réfléchir à propos d'un centre de calcul, peut-être de prendre des décisions, je ne sais plus.

Une discussion avait surgi, poussée sans doute par quelque hasard malin. La plupart des personnes présentes prétendant appartenir à la catégorie des vrais-informaticiens. Y compris l'une d'entre elles qui occupait, de fait, une fonction d'informaticien. Et cette dernière personne avait beaucoup de mal à réfuter les arguments qui lui déniaient la priorité d'appartenance au saint des saints. Et encore plus de mal à argumenter pour dénier l'appartenance des simples utilisateurs.

Cette scène qui remonte à bien des années m'amuse toujours, mais elle n'est que d'une manière indirecte à l'origine de mon propos. N'est-il pas normal qu'un personnage important rêve au moins une petite fois dans sa vie, voire tout au long d'une carrière, de laisser après son trépas, au beau milieu d'un grand carrefour, son image figée dans une attitude forte. Ou encore d'abandonner négligeamment à l'usage des foules passionnées, quelque chef d'oeuvre impérissable.

C'est ainsi que nous héritâmes d'un monument si terriblement abstrait, mais tellement formidable: la grandeur de la France. A l'instar du comédien de génie qui revêt comme on se joue mille personnalités, qui brille de mille éclats, un théatre versatile et divers nous échut de la même manière. Je ne m'en vais pas faire une liste exhaustive, la france est un pays trop riche. Mais le dernier en date de nos monuments s'élabore peu à peu, grandit comme une sorte de musique accordée sur l'infini. Il sera le plus fantastique. Jusqu'à présent nos trésors étaient jalousement gardés sur le terroir national. Celui-ci sera partout dans le monde. Même les petits africains du plus profond de la brousse, vont pouvoir

se gorger de son jus nourricier.

Enfin les africains et tous les enfants des espaces déshérités, repus de nourritures spirituelles, vont saisir toute la magie de l'être et du paraître.

Et comme nul n'est prophète en son pays, nous préparons ainsi la nouvelle génération des "vrais-informaticiens".

C'est en se basant également sur ce grand principe, car après tout pour l'instant il faut bien se contenter de ce qu'on a, que de nombreux informaticiens sont partis quelconques outre-atlantique et sont revenus auréolés de vrai. Mais tout s'use, et les charters aidant, cela ne fait plus guère exotique, aussi toujours plus haut, toujours plus loin, des esprits audacieux ont touché au Japon.

Et nous voila partis pour toute une croisade.

Alors me direz-vous, dans tout cela, les vrais-informaticiens, les faux-informaticiens? un vrai-informaticien peut-il être? un faux informaticien peut-il paraître?

A question question et demi: où un arbre vigoureux plonge-il ses racines? Dans une terre même ingrate, pourvu qu'un bon fumier l'enrichisse. Un vrai-informaticien surgira de lui-même d'un contexte convenablement enrichi; par de la psychologie?... par de la politique?...

En être ou n'en point être, l'éternelle question.

LES SYSTEMES INFORMATIQUES

par Patrick ISOARDI

Résumé : Cet article traite des Systèmes Informatiques.

Plus précisément, il présente les concepts de bases nécessaires à la réalisation de systèmes tels qu'ils sont conçus au sein du Laboratoire d'Informatique Théorique de Marseille-Luminy.

C.R Subjects Classifications Informatics 4-12 , 4-21 , 4-3 .

LES SYSTEMES INFORMATIQUES

Un D. E. A. d'Informatique appliquée à la conception et construction de processeurs et ordinateurs a été crée à la Faculté de Marseille-Luminy pour l'année universitaire 1982-83.

Dans le cadre de cet enseignement, placé sous la direction du professeur Edmond BIANCO, j'ai enseigné, pendant le mois d'octobre, les Systèmes Informatiques. Les étudiants admis à suivre ce D. E. A. proviennent d'horizons différents : Maîtrise d'Informatique, Maîtrise d'Automatique, Maîtrise de Physique option électronique, Maîtrise de Mathématiques avec certificats d'Informatique, ... La plupart d'entre eux sont déjà salariés et sont employés dans des entreprises liées à l'Informatique ou l'Electronique.
Il était donc nécessaire d'harmoniser les connaissances acquises et de définir un langage commun adapté à la notion des Systèmes Informatiques tels que nous les percevons au sein du Laboratoire d'Informatique Théorique de Luminy.

Je n'avais pas la prétention en quinze heures de cours de réaliser un système particulier, mais plutôt de fonder les concepts de bases nécessaires à la réalisation de tels Systèmes Informatiques.
Cet article retrace les points particuliers qui ont nécessité une mise à jour des connaissances acquises et plus généralement donne la définition d'un Système Informatique tel qu'il est abordé au sein du LITL.

I. NOTION DE SYSTEME.

Il n'est plus concevable d'utiliser raisonnablement un ordinateur sans un contexte logiciel relativement souple. C'est ainsi qu'un ordinateur possède un ensemble de compilateurs qui prennent en charge toutes les tâches introduites par les utilisateurs.
Que ce soit pour de la conversation, de la mise au point, ou simplement pour une bonne utilisation de la machine, tout programme doit pouvoir être interrompu. C'est là, qu'apparait la notion de système et plus particulièrement que se pose le problème de la distribution du travail aux

différents compilateurs.

Le problème fondamental ici posé, est celui du découpage des tâches, de la distribution d'une certaine quantité de déroulement à des portions d'algorithme.

On a deux catégories de solutions possibles :

1) La méthode des interruptions.

Les compilateurs n'ont pas à être structurés spécialement. Ils sont ce qu'ils sont dans leur description et on les interrompt au "bon moment" pendant qu'ils se déroulent ; ce qui revient à dire que le système les interrompt de force. Il se voit ainsi obligé de prendre des précautions de sauvegarde pour le travail dans lequel il intervient et qu'il lui faudra reprendre.

Très généralement, tout processus prioritaire se manifeste en interrompant ce qui se déroule. On a vu des systèmes ainsi bloqués par des trains d'interruptions trop rapprochés les uns des autres. De plus, la méthode n'étant pas toujours parfaitement définie, un grave problème se pose lors de l'introduction d'un nouveau compilateur au sein de ce système.

2) La méthode de distribution des tâches.

L'organisation des compilateurs fait partie intégrante de celle du système auquel ils appartiennent. Les compilateurs ne sont plus quelconques et tout se passe comme s'ils s'interrompaient eux-même au bon moment pour que le système puisse continuer. Ceci fait disparaître la notion d'interruption aléatoire ou introduit si on veut une interruption descriptible formalisable.

Les travaux du système et des compilateurs deviennent parfaitement séparés. Les compilateurs sont préstructurés de telle sorte que le système fonctionne comme un programme dans lequel s'insèrent des sortes de procédures dont le temps de déroulement, point important dans cette méthode, est strictement borné. Le travail complet étant réalisé par l'insertion d'autant de procédures qu'il est nécessaire. Le système ne peut plus être bloqué.

C'est la deuxième méthode qui est préconisée par les chercheurs du Laboratoire d'Informatique Théorique de Marseille-Luminy. Ce sont les concepts et structures nécessaires à une telle réalisation qui ont été exposés aux étudiants du D. E. A. et dont je vais ici présenter un résumé.

II. LE COMPILATEUR AUTOJECTIF.

1) Fini borné, fini illimité.

Pour que les choses marchent selon le principe de la dernière solution ci-dessus définie, il faut qu'une condition précise soit vérifiée : les compilateurs doivent être, subdivisés en éléments finis bornés dans le temps et dans l'espace mémoire.

Le champ informatique n'étant pas toujours celui du fini borné il convient de le diviser en deux domaines : le fini borné domaine de travail du compilateur ; le fini illimité pris en charge par le système.

2) L'autojectivité.

C'est une structure de logiciel qui permet la description complète de notre système informatique au regard des contraintes que l'on s'est précédemment imposées.

L'algorithme d'une procédure autojective est tel qu'il possède un seul point d'entrée et un seul point de sortie. De plus, il est structuré de telle sorte que tout déroulement qui emprunte le point d'entrée décrit un parcours fini borné qui l'emmène au point de sortie.

Le parcours est, pour faire image, une promenade sur un chemin. Le chemin est le support matériel, la promenade une sorte de déplacement sur ce support que rien n'empêche de parcourir plusieurs fois. En général, pour un même chemin, il existe plusieurs promenades dont le nombre dépend des arguments de la commutation : un nombre fini borné d'instructions déroulables, peut engendrer un nombre illimité d'instructions déroulées.

Il ressort alors de façon évidente, que la notion d'autojectivité commence par la notion de boucle : en effet, dans un algorithme,

les éléments qui sont susceptibles de ne pas être finis bornés dans le temps sont les boucles. Le fait de dégager la boucle en même temps que les paramètres qui la contrôlent donne le moyen d'obtenir une séparation de la commutation et du traitement. L'algorithme présente ainsi des points privilégiés au niveau desquels il suffit d'ouvrir la boucle pour la faire se refermer sur le système. Le système peut facilement compter les tours de boucles puisqu'il contrôle la commutation globale : il prend ainsi en compte la boucle à priori illimitée.

En conséquence on voit apparaître une partie des relations qui existent entre le fini-borné et le fini illimité et qui sont matérialisés entre les compilateurs et le système.

Propriété 1

Au sein d'un système de distribution des taches, les compilateurs seront nécessairement autojectifs.

La propriété est énoncée, il convient maintenant d'en user. Une technique de construction d'algorithmes autojectifs et présentée avec les COMPILATEURS COMPACTS (cf : Bulletin d'Informatique approfondie et applications n°2 pages 56-57-58).
J'ai moi-même défini une méthode de transformation d'un algorithme quelconque à une forme autojective (cf : Structures de Commutation, Thèse de 3ème cycle).
Ces études nous ont en outre permis de dégager des critères généraux de construction d'un bon lanage de programmation en égard de la manière dont est décrite la commutation. Le langage est prêt : il reste à le tester et à le publier.

III. LE SYSTEME DE DISTRIBUTION DES TACHES.

Nous avons défini la structure des compilateurs dans un système de distribution des taches.
Ces compilateurs sont autojectifs : chacun d'eux est succeptible d'effectuer un quantum de travail pour un utilisateur quelconque de telle manière qu'il puisse reprendre la tâche ainsi suspendue, au bout d'un délai

indéterminé mais fini et exactement au point où il l'avait laissé.

1) Les tâches.

A l'intérieur du système il existe une sorte de hiérarchie de la fonction. Le programme du système distribue les quanta de travail aux divers compilateurs qui sont de ce point de vue, tous au même niveau. Les compilateurs des langages de programmation confectionnent des piles de données ; la phrase ainsi formée est destinée à l'usage du compilateur de déroulement.
Le compilateur de déroulement travaille sur le programme généré et remplit des piles de résultats à usage cette fois-ci de l'utilisateur.
Le traitement des échanges est centralisé par le système.

Mais le rôle du système ne se limite pas à cela. Il doit en particulier assurer une certaine comptabilité des durées et implantations des diverses tâches, procéder aux distributions du travail en fonction des priorités établies, assurer la répartition de la place en mémoire ...

Il apparaît qu'il existe trois grandes catégories de tâches que rassemble le programme système et qui sont :

*) Le travail qui s'effectue sous le contrôle des compilateurs,
*) Le travail de gestion et de comptabilité,
*) Le traitement des échanges.

2) L'insertion des compilateurs.

L'insertion des compilateurs dans le système est un peu différente de l'insertion de procédure qui a été définie dans le cours de CESU et récemment présentée dans le Bulletin d'Informatique fondamentale et applications n°3.
L'insertion, les propriétés des langages de description des compilateurs et les choix de structure de la configuration des données sont liés.
Enoncer ces propriétés revient à définir le déroulement de l'insertion.

Propriété 2

L'écriture du compilateur est figée et son code est en exemplaire unique dans la mémoire.

Il doit pouvoir s'appliquer sur tout un ensemble de configurations différentes pourvu qu'elles soient homologues.

Définition

On dit que deux configurations sont homologues si elles diffèrent par les noms des cases mémoires et les contenus de ces cases, mais si les types des variables représentées par les cases se correspondent deux à deux.

C'est la désignation des opérandes qui fait le joint entre ces deux aspects : algorithme et configuration des données.

Propriété 3

Dans le langage utilisé pour la description des compilateurs, les opérandes sont nécessairement calculés par rapport à l'origine d'une configuration de la classe d'homologie définie pour le travail d'un compilateur.

Le compilateur peut être utilisé pour plusieurs tâches différentes. Il suffit de lui fournir, au moment de l'insertion, la bonne adresse pour qu'il use de sa configuration. Il s'agit là d'une question de système.

Propriété 4

L'origine de la configuration est un paramètre pour le compilateur. Cette adresse est fournie par le système qui gère la commutation globale.

Le compilateur doit pouvoir délaisser son travail et le reprendre ultérieurement là où il l'a laissé.

Propriété 5

La configuration attachée à un compilateur n'existe qu'à partir du moment où commence le premier déroulement de celui-ci pour une tâche donnée.
Elle subsiste tant que la tâche complète n'est pas achevée.
Elle n'existe plus, n'a plus de sens, à partir du moment où s'est terminé le dernier déroulement du compilateur pour la tâche en question.

Une telle méthode d'insertion n'est pas adaptée à la récursivité : la gestion des configurations en serait très compliquée. Mais quel intérêt de construire des compilateurs récursifs ?

Comme dans l'insertion de la procédure, on conserve là la pagination de la mémoire, bien adaptée au compilateur ; Et non pas ce découpage en tranches égales, trop rigide et indépendant de l'algorithme.

3) Gestion et comptabilité.

Les diverses fonctions de gestion et de comptabilité ne peuvent pas être imposées une fois pour toutes. Il est possible de remplacer dans le programme du système des procédures par d'autres dont on estime qu'elles sont plus adaptées à la situation du moment. Nous verrons ultérieurement le moyen de réaliser cette transformation.
Toutefois, pour que cette modification soit possible, il faut que le système possède dans sa description initiale une structure de logiciel adaptée à l'insertion des procédures.

4) Les paramètres de la commutation.

La distribution des tâches impose que lors de l'abandon d'une tâche, le système a nécessairement besoin d'informations suffisantes qui indiquent sans ambiguité à quel point la tâche doit être reprise.
Ces informations doivent être présentent dans la mémoire.
Je ne développerais pas ici dans le détail toutes les catégories d'informations nécessaires à la communication : elles dépendent du problème choisi.
Par contre, une propriété reste commune :

Propriété 6

La structure ou les statuts de chaque catégorie d'informations seront déclarés dans la description initiale du système.
La dimension de la partie de la mémoire succeptible de contenir des informations d'une même catégorie y sera également définie.

Ceci nous ramène à la description de files par leurs statuts et leur dimension ; Files périodiques pour la liste des distributions de tâches ou la liste des attentes, ... ; Files apériodiques pour les données ou les programmes.

5) La gestion de la mémoire.

Nous avons décrit le système du point de vue de l'imbrication et de la structure des programmes qui le composent.
Nous avons décrit la partie déclarative commune des paramètres de communications entre les différents programmes.
Il nous reste, pour achever la description du système, à montrer comment il est possible de gérer le déroulement de l'ensemble des programmes utilisateurs ainsi que l'occupation de la mémoire.

Les programmes constitués d'une suite de procédures ou de structures équivalentes ont une géométrie invariable en cours de déroulement. Les données, au contraire, apparaissent pendant le déroulement et leur volume est succeptible de fluctuer fortement.
Pour gérer ces fluctuations, il est possible d'opérer de plusieurs manières : par exemple

- on divise la zone disponible de la mémoire en plusieurs parties : égales ou non
- la zone disponible n'a que deux parties : la partie occupée et la partie libre que l'on remplit lorsque les besoins se présentent et qui se vide en laissant des espaces inoccupés à recondenser.

Il semble difficile de décider d'un choix de partage de la mémoire ; mais je peux imposer la propriété suivante :

Propriété 7

Il est préférable de construire un système dans lequel, initialement une méthode simple de répartition de la place mémoire est programmée, mais dont la programmation prévoit de modifier la fonction de répartition.

Ce moyen que nous donnons est efficace pour mener à bien cette gestion de la mémoire et laisse une grande liberté à l'opérateur.

6) Le traitement des échanges.

Il paraît que la gestion des entrées-sorties est l'une des tâches les plus difficiles à envisager qui soient dans la réalisation des systèmes. Cela tient essentiellement à la complexité et à la diversité des mécanismes utilisés pour ces échanges d'informations.

Si on ne retient au niveau du système que les propriétés communes

à ces dispositifs, alors le traitement des échanges se place au même niveau que tout autre traitement.

Propriétés communes

▽ Il existe une variable accouplée à la file externe du dispositif. Cette variable sert à l'échange de renseignements entre le système et le dispositif.

▽ La file externe est supposée autonome. On lui fait correspondre une file interne de structure homologue.

Il est bien connu qu'un automate est suffisant pour décrire le traitement des échanges. La variable associée est une image des états de l'automate : "attente", "exécution", "terminé", ...

En quelques mots, le processus est le suivant :

- La variable associée à la valeur "attente".
- Le système effectue la commande d'entrée ou de sortie.
- Il vérifie qu'elle est acceptée : la variable associée doit prendre la valeur "exécution".
- Lorsque la variable associée prend la valeur "terminé", la file interne ou son contenu peuvent être utilisés par le système.

On a ainsi un programme "canal". Ce programme sera lu par le calculateur d'échanges.

7) Le calculateur d'échanges.

Le transfert de l'information est ce qui est de plus coûteux, en temps, dans un calculateur.

Afin d'utiliser au maximum le travail de l'unité centrale pour les calculs et de rendre le traitement des échanges le moins coûteux possible, nous allons énoncer les propriétés nécessaires et suffisantes que doit posséder un calculateur d'échanges.

Propriétés

▽ Le calculateur d'échanges fonctionne de manière indépendante après qu'il ait reçu les informations indispensables :

- Adresse du dispositif
- Adresse de la file interne
- Type de l'échange

▽ Le calculateur travaille dans la mémoire centrale au même titre que l'unité centrale

- Il lit le programme "canal" qui lui est fourni par le système
- Il note ses états successifs dans la variable associée
- Il provoque le transfert des données entre la file externe et la file interne homologue.

Conséquences

Après avoir lancé l'échange, l'unité centrale travaille en véritable simultanéité avec les différents calculateurs d'échanges.
Nous avons ainsi réalisé la séparation fondamentale qui existe entre :
- le traitement des données, travail rapide effectué par l'unité centrale
- le transfert des données, travail lent effectué par les calculateurs d'échanges.

IV. LE LOGICIEL.

Les langages de programmation sont commodes et normalement suffisants pour répondre aux besoins des utilisateurs qui ne sont pas tous des informaticiens.

1) Les langages de correction.

Il faut compléter le manque de redondance naturel du langage de programmation à l'aide d'un langage spécialisé qui permet la correction à posteriori.

Il est indispensable, la tâche étant bien avancée ou même terminée de pouvoir revenir sur une faute passée inaperçue ou préalable ; ou même de pouvoir reprendre des passages de programmes que l'on veut changer ou améliorer.

Les corrections que l'on peut apporter à une phrase d'un langage dépendent de ce dernier. C'est ainsi que nous aurons dans notre système un langage de correction pour chacun des langages de programmation.

2) Les langages de traitement des données.

L'opération qui consiste à introduire ou à extraire une valeur de la mémoire centrale est différente de celle qui consiste à traiter cette valeur

au sein même de la mémoire.

Il semble naturel de traiter cette opération de manière indépendante. Un langage est donc prévu d'une part pour constituer les stocks de données dans lesquels les programmes puisent des valeurs initiales et entassent des résultats et d'autre part pour communiquer avec l'extérieur.

Là encore le type de l'échange étant fonction du langage de programmation, nous aurons un langage de traitement de données pour chacun des langages de programmation.

3) Le langage hors-texte.

L'utilisateur qui veut résoudre un problème dispose de tout un choix de langages parmi lesquels il fait appel à celui qu'il estime le plus adapté à ses préoccupations.

Le langage doit être disponible chaque fois qu'un terminal est en état d'attente.
L'utilisateur qui désire alors travailler déclare, outre son nom et son code, le nom du langage qu 'il utilise.
Ceci doit se faire dans un langage différent de tous les précédents mais qui est fonction des précédents : les noms des langages font partie de l'alphabet de ce nouveau langage.
C'est ce langage unique que nous appelons le langage hors-texte.

4) Le langage de communication.

La réalisation d'un système est un travail énorme.
Un centre de calcul peut être modifié : apport d'un nouveau langage ou d'un périphérique supplémentaire.
La nature de cette transformation est généralement simple, mais la compléxité des systèmes est telle que, le plus souvent tout ou partie de ces derniers doit être reécrit.
Nous allons étudier les outils nécessaires pour que cette transformation soit naturelle et ne modifie en rien l'écriture initiale du système.

Il va de soi que rajouter un langage de programmation implique de rajouter en même temps le langage de correction et le langage de traitemer

des données qui lui sont associés, ainsi que leurs noms dans l'alphabet du langage hors-texte.
L'introduction dans le système des compilateurs (bien entendu autojectifs) qui correspondent à ces langages se fait par l'intermédiaire d'un langage de communication dont les propriétés sont les suivantes :

Propriétés

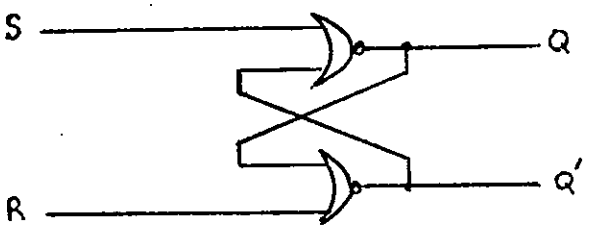
Ce nouveau langage permet de modifier le fonctionnement du système
∇ en rajoutant les statuts des paramètres de commutation attachés aux nouveaux compilateurs, dans la liste initiale.
∇ en insérant en des points (déterminés à l'avance) du corps de programme du système, des nouvelles procédures.

Le problème est actuellement à l'étude.
La solution est proche.

QUELQUES REMARQUES SUR UN SYSTEME SEQUENTIEL

J.Ph.Lehmann

Pour illustrer notre propos, nous reproduisons ici des extraits significatifs d'un texte fort connu, paru il y a quelques années dans la revue "L'informatique". Il s'agit de présenter un circuit élémentaire, en l'occurrence une bascule du type R/S. Ces circuits interviennent en logique séquentielle, précisément en matérialisant la fonction de mémorisation.
Voici les extraits en question:



NOR

e_1	e_2	S^x
0	0	1
0	1	0
1	0	0
1	1	0

" Pour voir ce que donne le système, les sorties Q' et Q influant sur les entrées, faisons des hypothèses à leur sujet: imaginons qu'à un instant t, R=0, S=0, et Q_t=0.
Dressons progréssivement la table de vérité du système à l'instant suivant. .....
De manière générale on voit que Q' ne peut valoir que $\overline{Q}$.
.....
Essayons Q_t=1 Q'_t=0, ...Essayons R=1 S=0 Q_t=1 .....
.....
Lorsque R=S=1; le système devrait sortir soit simultanément Q=Q'=0, soit Q=Q'=1, ce qui est impossible..... "

On voit bien la méthode d'analyse qui est proposée: on fait des

hypothèses sur l'état des éléments hors de notre contrôle, et on en tire un certain nombre de conclusions: si on aboutit à des états stables, alors on retient l'hypothèse comme valide. Or malheureusement le seul cas où ces hypothèses pourront être, soit écartées, soit rejetées, se présentera justement, soit lorsque l'état engendré aura un caractère instable, soit lorsque des contradictions se présenteront.
Par ailleurs ces considérations de nature purement logique, nous mènent, dans le cas où nos hypothèses n'entrainent aucune contradiction, à ne pouvoir rien dire du tout. Nous ne savons pas, en effet, si ces hypothèses sont des thèses, c'est à dire vraies, et de ce fait rien sur la vérité des conclusions ne pourra être énoncé.

Aussi suggérons-nous d'aborder ce type de problèmes d'une façon différente:

- de tels systèmes se présentent avec des entrées, dont l'état est en fait inconnu. Il n'est donc pas correct d'envisager certaines hypothèses, mais l'ensemble de celles-ci. Si alors l'évolution du système aboutit à définir un état stable, la question de savoir quelle est, en fait, l'hypothèse qui l'a engendré, devient inessentielle, et en outre la conclusion se trouve indiscutablement avérée.

- D'autre part, il nous semble indispensable d'étayer nos démonstrations en partant de suppositions qui portent au même instant, à la fois sur Q et sur Q'. En effet,en procédant autrement, nous ne pourrions, au mieux, qu'aboutir à des conclusions sur une pseudo-stabilité des états, dans la mesure où, entre chaque transition associée à un état connu, et la suivante, se situent d'autres transitions tout à fait inconnues. Par exemple R=1, S=0, Q_t=0, mènerait à Q'_{t+1}=0, Q_{t+2}=1, mais nous n'aurions aucun renseignement ni sur Q_{t+1}, Q_{t+3},.. ni sur Q'_t, Q'_{t+2}, etc....

-Enfin nous continuerons à supposer que la "discrétisation" des processus de transition dans le temps est licite: en réalité la questionde savoir si cette methodologié est correcte, ne se pose vraiment que dans le cas où les états engendrés sont instables, ce qui ne pourrait donner lieu qu'à de bonnes surprises. Dans les autres situations, cette convention est bien moins critiquable, dans la mesure où; quelque soit l'état des entrées à un instant donné, on aboutit tout de même au même état de sortie.

Examinons maintenant les conclusions entrainées par la méthode que nous avons dégagée:

	Q_t	Q'_t	Q_{t+1}	Q'_{t+1}	Q_{t+2}	Q'_{t+2}
R S = 0 0	0	0	1	1	0	0
	0	1	0	1	0	1
	1	0	1	0	1	0
	1	1	0	0	1	1
R S = 0 1	0	0	0	1	0	1
	0	1	0	1	0	1
	1	0	0	0	0	1
	1	1	0	0	0	1
R S = 1 0	0	0	1	0	1	0
	0	1	0	0	1	0
	1	0	1	0	1	0
	1	1	0	0	1	0
R S = 1 1	0	0	0	0	0	0
	0	1	0	0	0	0
	1	0	0	0	0	0
	1	1	0	0	0	0

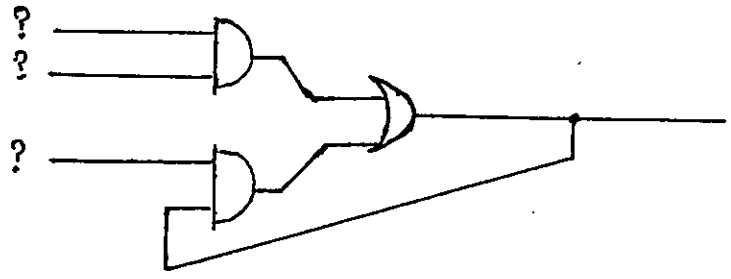
L'examen du tableau nous indique, qu'à l'exception de la commande RS=00, nous sommes assurés d'engendrer 3 états stables en sortie 01, 10, 00.
D'autre part la commande RS=00, n'est susceptible dengendrer des états instables qu'associée à une sortie 00 ou 11.
Dans ces conditions, il apparait que les recommandations visant à exclure la commande RS=11, parce qu'elle détermine des états instables, témoignent d'une vision inexacte du phénomène. En fait RS=11 définit un état stable de sortie 00. Alors?

La réponse que nous proposons est la suivante:
Si RS=11 doit être écartée, c'est au contraire parcequ-elle détermine de façon stable 00 en sortie, et qu'une telle sortie, si elle vient à être associée à la commande RS=00, engendrera l'état instable 00,11,00,11,...etc.
Compte tenu de cette restriction, il ne reste plus qu'à constater que le système fonctionne tout à fait comme il est dit dans les livres:

-Faisant RS=01 (set) Q' est mis à 1
Si RS retombe à 00 l'état de sortie est maintenu.
-Faisant RS=10 (reset) Q' est mis à 0
Si RS retombe à 00 l'état de sortie est maintenu

En outre on a constamment $Q' = \overline{Q}$.

En quelque sorte, l'occasion nous est ici offerte de nous réjouir de ce que le hasard fait bien les choses.
Le lecteur intéressé pourra s'exercer à approcher de la même façon l'étude du circuit suivant:
Comment utiliser les entrées pour en faire une mémoire?



Autojection et Compilateurs

suite

e. bianco

Lors des deux premiers articles, j'ai éssayé de rédiger une présentation globale du point de vue sous lequel j'aborde la compilation. Et j'ai tenté de donner une idée du cadre formel dans lequel je me suis placé. Un compilateur n'est jamais dans la réalité un objet isolé, il fait toujours partie d'un contexte où apparaîssent d'autres compilateurs, de la gestion d'échanges, de la gestion d'utilitaires, etc.

Il paraît ainsi tout à fait raisonnable de concevoir une tâche centrale de répartition de la mémoire et du temps de l'unité centrale. Cette tâche est plus habituellement rassemblée sous le vocable: système de gestion de l'ordinateur, ici plus simplement: système.

Tout acte de compilation se fait dans le but essentiel d'aboutir au déroulement du programme construit. Ceci se représente en deux phases, qui peuvent d'ailleurs se mélanger,

l'une, dite phase de compilation a pour but de construire ce qui va se dérouler dans la seconde.

L'autre, donc, est la phase de déroulement.

A l'origine du traitement on place le programme de l'utilisateur,, l'usinage fournit un produit qui est l'image de ce programme, et suscentible d'être déroulé afin d'obtenir le calcul, but ultime de toute l'opération.

Pendant que se déroule un tel processus, des programmes et des configurations de données sont étalés en mémoire. Je vais donner une image de ce dont il faut disposer dans chacune de ces phases. Les points de suspension me servent à séparer les paquets d'information indépendants.

Pour la première phase:

..{(code du compilateur)..{(n1)(paramètres(aΨ),(aΨ),(aER)(variables locales: (variables d'état),(tableau de variables),(n2))..{(Ψ)...
d'adresse x d'adresse L_I d'adresse (aΨ)

{(Ψ) ...
d'adresse (aΨ)

La première ligne décrit les objets et la ligne juste en-dessous, leurs adresses. Par exemple, l'adresse "x" du compilateur n'est utile que pour le système.

- (n1) représente les premières cases qui jouent un rôle organique dans toute configuration de procédure.
- (n2) représente tous les paramètres autres que ceux qui sont précisés.
- (n3) toutes les variables locales autres que celles qui sont également précisées.
- (Ψ) représente la phrase symbolique à traduire,
- (Ψ) représente la phrase image de la précédente après traduction.

Pour la seconde phase:

..{(code du compilateur de déroulement)..{(n1),(paramètres(aΨ),(aD),
d'adresse y d'adresse L_I

(ER),(variables locales du compilateur)... ( )... ...
d'adresse aΨ/ Γ_i

(données de)... (n/),(paramètres/d),(variables locales/d)... ...
d'adresse dD/Γ_i d'adresse d/Γ_i

Je me place dans le cas le plus général lorsque je prévoie un compilateur de déroulement.C'est la situation décrite ci-dessus.Mais on peut concevoir que dans certains cas simples,l'unité centrale puisse jouer ce rôle.Alors il n'y a plus de code du compilateur en mémoire,et sa configuration disparaît dans la plupart des cas à l'intérieur du processeur.

J'insiste sur le fait que nous voyons apparaître maintenant la configuration sur laquelle s'applique Ψ,elle n'existait pas dans la première image. J'entends par là qu'elle n'occupait aucune place en mémoire.

DESCRIPTION DU COMPILATEUR.

Préambule.

La seule description rigoureuse ne peut se faire qu'avec un "bon" langage de programmation.Le PFS,par exemple.Mais si ce langage offre bien des commodités pour la rédaction d'un objet de cette nature,il conserve néanmoins, comme tous les langages synthétiques proches des ordinateurs,un aspect rugueux qui n'en facilite pas la lecture.

Je vais donc essayer d'utiliser plutôt un langage à base de prescriptions plus proche du français courant,et ainsi me semble-t-il,plus facilement abordable pour l'esprit humain.

Je conserverai l'armature telle que je l'ai rédigée en PFS,puisque mon intention de départ était de présenter le compilateur sous cette forme.

- 1 - DECLARATION DES VARIABLES

Ce sont des files du PFS qui me servent à déclarer les variables du compilateur:

Commentaire Tableau des variables d'état:

```
1( pro(1) , decl(1) , aff(1) , cond(1) , inser(1) , vers(1) , ident(1) ,
        par(1) , nb(1) ): synt ;
```

Commentaire Variables locales diverses dont la signification est explicitée plus bas.

```
1( type(1) , dim(2) , no(2) , npe(2) , nb(2) , nvvl(2) , op(1) , num(2) ,
al(2) , ai(2) , vp(2) , ctrl(2) , type2(1) , dim2(2) , no2(2) , locp(2) ,
maxef(2) , nos(2) ): vl ;
```

Commentaire File destinée à contenir le programme généré:

```
1 000 , 3000 , 1 ( car(1) ) ,
        2 ( adr(2) ): g / p ;
```

Commentaire File qui contient la phrase symbolique à compiler:

```
500 ( car(1) ): let ;
```

et la file qui contient l'erreur: `1(car(1)):err`

Commentaire File destinée à contenir le tableau des variables telles qu'elles sont déclarées dans la phrase symbolique.

```
50 ( ident(4) , type(1) , dim(2) , no(2) ):ti : tf : tc : tx ;
```

Commentaire File qui sert à construire l'identificateur et à le mettre en réserve (variables indicées).

```
1 ( 4(1): l ): h ;          1( 4car(1) ): h2 ;
```

Méta-commentaire Il me semble utile avant de commencer la description, de rappeler le rôle que jouent les variables locales diverses:

- no Numéro de la variable déclarée. Il est incrémenté à l'apparition

de chaque variable.

- npe — décomptage des paramètres effectifs.
- nvvl — nombre de cases occupées par les variables locales.
- num — numéro d'opérande sur lequel on se trouve.
- vp — volume du programme généré.
- locp — localisation de procédure.
- maxef — nombre maximum de paramètres effectifs.
- ctrl — compteur de lettres.
- nos — réserve le nombre de variables déclarées. Sert notamment à localiser les cases de manoeuvre utiles au déroulement.

Fin du méta-commentaire.

Je me donne maintenant le jeu des valeurs des variables d'état, c'est à partir de ce jeu que je construis l'analyseur syntaxique.

pro	0	1	2	0
lettre		proc	début	fin

decl	0	1	2	3	0
lettre		par	vl	index	début

aff	0	1	0	2	0
lettre		:=	;	Ⓐ	;

cond	0	1	2	3	0
lettre		si	Ⓡ	vers	;

inser	0	1	2	3	0
lettre		inser	(	)	;

vers	0	1	0
lettre		vers	;

ident	0	1	1	0
lettre		lal	lal_{ch}	;

par	0	1	2	0
lettre		(	)	:= , vers ; Ⓐ Ⓡ

Commentaire Le mot "lettre" désigne bien sûr, la lettre de l'alphabet du langage. Les notation: "lal", "ch", A , R ,désignent successivement: les lettres de l'alphabet latin, les chiffres de 0 à 9 , les opérateurs arithmétiques, les opérateurs de relation.

- 2 - AIGUILLAGE

L'algorithme du compilateur commence comme nous l'avons vu, par un aiguillage. Celui-ci renvoit dans les algorithmes de traitement des classes syntactico-sémantiques de lettres. Chacun de ces algorithmes qui porte le nom de sa classe se termine en un point qui le renvoit à la fin commune que je désigne précisément par l'étiquette: "fin commune".

Aiguillage

Quand la lettre à étudier qui se trouve en L_{let} est:	Alors on déroule l'algorithme:
procédure ou pro	procédure
paramètre ou par	paramètres
variable locale ou vl	var-loc
index ou ind	index
début ou deb	début
fin	fin
insérer ou ins	insérer
si	condition
vers	aller-à
;	point-virgule
,	virgule
:=	deux-points-égal

:	deux-points
(	par-ouvrante
)	par-fer
+ - . /	op-arith
= < > ≥ ≤	relations
A B C D ... Z	lettres
0 1 2 3 ... 9	chiffres

- 3 - CLASSES DE LETTRES

Je décris alors les algorithmes qui traitent de chacune des classes syntactico-sémantiques de lettres, définies à partir du rôle que jouent ces lettres dans la phrase. Je vais user de prescriptions qu'il suffit de suivre à la lettre pour réaliser le calcul. Cette description est faite tout spécialement pour un calcul "à la main". Mais comment saisir tout le jeu subtil de la compilation si l'on n'a jamais fait ça au moins une fois dans sa vie ?

procédure

A) cette lettre est rencontrée hors programme (<nro> = 0):

<u>syntaxe</u> (<pro> passe à "/")

<u>sémantique</u> (mettre 3 en <no> , 1 en <nvvl> , 0 en <vp> ,
0 en <ctrl> , 0 en <dim>,
0 en <npe> , 3 en <maxef>,
mettre à blanc l'identificateur en <h> ,
<u>procédure</u> geninstl (qui génère la mise en place du volume en case 0 de la configuration au déroulement.)
<u>vers</u> fin commune

paramètres

A) En déclaration (<decl> = 0):

syntaxe (<decl> passe à "1")

sémantique (<type> passe à "par") ; vers fin commune;

B) sinon (<decl> ≠ 0):

sémantique (<err> passe à "er1"):vers fin commune

---

var-loc

A) En déclaration (<decl> = 1 , 0):

syntaxe (<decl> passe à "2")

sémantique (<type> passe à "vl"):vers fin commune;

B) sinon (<decl> =2 ou<pro> ≠ 1):

sémantique (<err> passe à "er2"):vers fin commune.

---

index

A) En déclaration (<decl>=0 , 1, 2):

syntaxe (<decl> passe à "3"):

sémantique (<type> passe à "ind"):vers fin commune;

B) sinon (<pro> = 1):

sémantique (<err>passe à "er3"):vers fin commune;

Commentaire Comme on peut le constater,chaque entrée de paragraphe est une condition;quand celle-ci n'est pas vérifiée,il faut passer au paragraphe suivant.S'il n'y a pas de paragraphe suivant cela signifie simplement que l'algorithme est sujet à caution.On peut constater également que chaque sémantique se termine sur un vers fin commune,je supprimerai donc cette dernière

indication, qui est visiblement superflue.

Commentaire 2 La désignation de la variable x, par exemple, sous la forme x, est une abréviation pour $\mathcal{L}_{y,x}$. Cette écriture n'est pas trop ambigüe dans la mesure où les identificateurs utilisés dans les files de références synt et vl, sont tous différents. Or, ici ce n'est pas le cas. Mais heureusement un critère supplémentaire permet de les discerner. Les uns sont utilisés en rubrique syntaxe exclusivement, et les autres en rubrique sémantique.

---

début

A) Dans le cas où <pro> = "/" :

syntaxe (mettre "2" en <pro>, et "0" en <decl>)

sémantique (mettre en conserve <no> en <nos> (nécessaire pour connaître les noms des 3 cases de manoeuvre de la configuration au déroulement.))

B) sinon (<pro> = "/"):

sémantique (mettre "er4" en <err>)

---

inserer

A) vérifie que <ident> = "0", <par> = "0", <pro> = "0":

syntaxe (mettre <inser> à "/"): vers fin commune;

B) sinon : sémantique (mettre "er5" en <err>);

---

deux-points

A) En instructions (<pro> = "2"):

A1) ferme un identificateur (<ident> = "/"):

syntaxe ("0" en <ident>)

sémantique (mettre "étiq" en <type>, mettre <nvvl> en <no>,

proc charvar (qui envoie la variable en table de variables);

A2) sinon: sémantique (mettre "er6" en <err>);(mauque l'étiq.)

B) sinon : sémantique (mettre "er7" en <err>);

---

deux-points-égal

A) En instructions (<pro> = 2): sémantique (mettre 0 en <num>):

A1) Cas où (<aff> , <cond> , <ins> , <vers>) = "0" :

A11) ferme un identificateur (<ident> = "/"):

syntaxe(<ident> à "0" ,<aff> à "/")

sémantique (proc rechvar :

A111) Il n'y a pas erreur (<err> = "0"):

sémantique (proc genop (génère l'instruction qui correspond à l'opérande simple.)

A112) Il y a erreur: vers fin commune;

A12) Suit une ")" (<par> = "/"):

syntaxe (<par> à "0" , <aff> à "/")

sémantique (proc genopind,blanc en <h> (identif.), 0 en <ctrl>);(genopind,génère le jeu d'instructions de l'opérande indicé.)

B) sinon (<pro> ≠ "2"):

sémantique (mettre "er8" en <err>);

---

lettre

A) Hors identificateur (ident> = "0"):

sysntaxe (mettre "/" en ident);

sémantique (écrire la lettre à la suite en <h> ,ajouter 1 en <ctrl>

B) Dans un ident. (ident> = "/"): sémantique(B1) pour <ctrl> = 4: err := "er9": fin.

B2) lettre à la suite, + 1 en ctrl)

chiffre

A) Dans un identificateur (ident>=1): sémantique (écrire à la suite en <h>, ajouter 1 en <ctrl>)

B) Dans un nombre (nomb>= 1): sémantique (valeur du chiffre ajoutée à <nb> X 10);

C) Sinon pour (nomb ="0"):

C1) Si entre-parenthèses (par>="1"): syntaxe (mettre "1" en <nomb>)
sémantique (valeur du chiffre ajoutée à <nb> X 10)

C2) Ou bien pour (inser>= 2), ou pour (aff>="1") ou pour (cond>="1", 2):
syntaxe (mettre "1" en <nomb>)
sémantique (mettre <nb> à 0, ajouter la valeur du chiffre à <nb>)

---

op-arith

A) En affectation (<aff> = "1"): sémantique (mettre 1 en <num>):

A1) Quand on a "+" sémantique (mettre ":=+" en <op>):

A2) Quand on a "-" sémantique (mettre ":=-" en <op>):

A3) Quand on a "*" sémantique (mettre ":=*" en <op>):

A4) Quand on a "/" sémantique (mettre ":=/" en <op>):

AA1) Fermant d'identificateur (ident>="1"):

syntaxe (mettre "0" en <ident>, mettre "2" en <aff>)

sémantique (proc rechvar, proc genop, mettre 0 en <ctrl>, mettre l'identificateur à blanc)

AA2) Suit une ")" (<par> ="2"):

syntaxe (mettre "0" en <par>, et "2" en <aff>)

sémantique (proc genopind, mettre 0 en <ctrl>, mettre l'identificateur à blanc)

AA3) Autres cas, erreur: sémantique (mettre "er/0" en <err>);

B) Autres cas: erreur: sémantique (mettre "er/1" en <err>);

---

virgule

A) En liste de paramètres (<decl> = "1"):

A1) Ferme un identificateur (<ident> = 1):

syntaxe (mettre "0" en <ident>)

sémantique (proc charvar :

A11) En cas d'erreur : vers fin commune;

A12) Sinon: ajouter 1 en <no> , mettre <ctrl> à 0,

mettre l'identificateur à blanc)

A2) Sinon: erreur: sémantique (mettre "er/2" en <err>);

B) En liste de variables locales (<decl> = "2"):

B1) Ferme un identificateur (<ident> = "1"):

syntaxe (mettre "0" en <ident>)

sémantique (proc charvar :

B11) En cas d'erreur: vers fin commune);

B12) Sinon: mettre <ctrl> à 0 , proc adrvl , + 1 en <nvvl> ,

blanc en identificateur , + 1 en <no>);

B2) Suit une ")" (<par> = "2"):

syntaxe (mettre "0" en <par>)

sémantique (ajouter <nb> en <nvvl> , Proc adrvl , ajouter 1 en <no>,

mettre 0 en <dim>)

C) En liste d'index (<decl> = "3"):

C1) Ferme un identificateur (<ident> = "1"):

syntaxe (<ident> à "0")

sémantique (proc charvar :

C11) En cas d'erreur: vers fin commune;

C/2) Sinon: (proc advl ,ajouter / en <nvvl> , ajouter / en <no> ,
blanc en identificateur , 0 en <ctrl>);

C2) Sinon :erreur: mettre "er/3" en <err> ;

D) En instruction :

D/) En insertion (<inser> ="2"):

D/1) Ferme un identificateur:

syntaxe (mettre "0" en <ident>)

sémantique (proc rechvar , proc genepar ,ajouter / en <npe> ,
blanc en identificateur , 0 en <ctrl>);

D/2) Ferme un nombre (<nomb> ="/"):

syntaxe (mettre "0" en <nomb>)

sémantique (proc genct , mettre <nos> + 3 + <num> en <no> ,
proc genepar , ajouter / en <npe> et <num>);

D/3) Suit une ")" (<par> ="2"):

syntaxe (mettre "0" en <par>)

sémantique (proc genparind , ajouter 1 en <npe>)

E) Sinon : erreur : mettre "er/4" en <err> ;

---

point-virgule

A) En déclaration (<pro> = 1):

A/) Ferme le nom de procédure (<decl> ="0"):

A/1) Il y a un identificateur (<ident> = /):

syntaxe (mettre "0" en <ident>)

sémantique (placer ce nom en file des procédures ,mettre 0 en
<ctrl> , mettre à blanc l'identificateur);

A/2) Sinon : erreur : mettre "er/5" en <err> ;

A2) Ferme la liste des paramètres (<decl> ="/"):

A2/) Il y a un identificateur (<ident> ="/"):

syntaxe (mettre <ident> à "0")

sémantique (proc charvar :

A211) En cas d'erreur : vers fin commune;

A212) Sinon : ajouter 1 en <no> , <ctrl> à 0 ,blanc en identificateur ;

A22) Manque un identificateur : erreur : mettre "er16" en <err> ;

A3) Ferme la liste des variables locales (<decl> = "2"):

A31) Ferme un identificateur (<ident> = "1"):

syntaxe (mettre "0" en <ident>)

sémantique (proc charvar

A311) En cas d'erreur : vers fin commune ;

A312) Sinon : (proc advl , ajouter 1 en <no> , ajouter 1 en <nvvl> ,mettre <ctrl> à 0 , mettre blanc en ident);

A32) Suit une ")" (<par> = "2"):

syntaxe (mettre "0" en <par>)

sémantique (ajouter <nb> en <nvvl> , proc advl , ajouter 1 en <no> , mettre 0 en <ctrl>en <dim> ,blanc en identificateur)

A331) Sinon : erreur : mettre "er17" en <err> ;

A4) Ferme la liste des index (<decl> = "3"):

A41) Ferme un identificateur (<ident> = "1"):

syntaxe (mettre "0" en <ident>)

sémantique (proc charvar :

A411) En cas d'erreur : vers fin commune ;

A412) Sinon : (proc advl , ajouter 1 en <no> , 1 en <nvvl>, mettre 0 en <ctrl> , blanc en identificateur ,)

B) En instruction (<pro> = 2):

B1) Ferme une affectation (<aff> = "1" ou "2"): sémantique (mettre 2 en <num>):

B11) Ferme un identificateur (<ident> = "1"):

syntaxe (mettre "0" en <ident>, "0" en <aff>)

sémantique (proc rechvar :

A511) En cas d'erreur : vers fin commune);

A512) Sinon : proc genop , proc genaffim ,);

B12) Suit une ")" (<par> = "2"):

syntaxe (mettre <par> à "0"):

sémantique (proc genopind , proc genaffim , blanc en identificateur ,0 en <ctrl>)

B13) Sinon : erreur : mettre "er18" en <err> ;

B2) Ferme une condition (<cond> = "3"):

B21) Ferme un identificateur (<ident> = "1"):

syntaxe (mettre "0" en <cond> , "0" en <ident>)

sémantique (mettre "cond" en <type> , mettre <nvvl> en <no> ,

proc charvar1);

B22) Sinon : erreur : mettre "er19" en <err> ;

B3) Ferme un vers (<vers> = "1"):

B31) Ferme un identification (<ident> = "1"):

syntaxe (mettre <ident> à "0" , <vers> à "0")

sémantique (mettre "vers" en <type> , mettre <nvvl> en <no> ,

proc charvar1);

B32) Sinon : manque un ident. : mettre "er20" en <err> ;

B4) Ferme un inserer (<ins> = "3"):

syntaxe (mettre "0" en <ins>)

sémantique (proc geninsim :

B41) Si <num> > <maxef> : <num> remplace <maxef> , mettre 0 en

<noe> , 0 en <num>);

B42) Sinon : mettre 0 en <npe> , 0 en <num>);

Commentaire Il m'est nécessaire de préciser deux prescriptions qui servent fréquemment: "Ferme un identificateur" signifie qu'on vérifie que ident = 1 et que ctrl = 4 ; L'autre prescription: "Déplacer les caractéristiques" signifie: mettre <h> , <type> , <dim> , <no> en <h2> , <type2> , <dim2> , <no2> .Je rappelle également que c'est dans la file de référence <h> que je place l'identificateur , et dans la file d'identificateur <h2>, que je place son double.

---

par-ouvrante

A) En déclaration (<pro> = "1"):

A1) En variable locale (<decl> = "2"):

A11) Ferme un identificateur (<ident> = "1"):

syntaxe (mettre "0" en <ident> , <par> passe à "1")

sémantique (mettre 0 en <nb>);

A12) Sinon : erreur : mettre "er21" en <err> ;

A2) Sinon : erreur : mettre "er22" en <err> ;

B) En instructions (<pro> = "2"):

B1) Suit le nom de procédure (<inser> = "1"):

B11) Ferme un identificateur (<ident> = "1"):

syntaxe (mettre "0" en <ident> , et "2" en <inser>)

sémantique (proc rechproc , mettre sa distance en <locp> ,

blanc en identificateur, mettre o en <ctrl> et <num>)

B12) Sinon : erreur : mettre "er23" en <err> ;

B2)

B21) Ferme un identificateur (<ident> = "1"):

syntaxe (<par> passe à "1", <ident> passe à "0")

sémantique (proc rechvar

B211) En cas d'erreur : vers fin commune ;

B212) Sinon : déplacer les caractéristiques ,blanc en identificateur , <ctrl> à 0);

B3) En condition (<cond> = "1", "2"):

B31) Ferme un identificateur (<ident> = 2):

syntaxe (mettre "0" en <ident> , et "1" en <par>)

sémantique (proc rechvar :

B311) En cas d'erreur : mettre "er24" en <err> ;

B312) Sinon : déplace les caractéristiques ;

B4) En aller-à (<vers> = "1"): erreur : mettre "er25" en <err> ;

B5) En affectation (<aff> = "0", "1", "2"):

B51) Ferme un identificateur (<ident> = "1"):

syntaxe (mettre "0" en <ident> ,et "1" en <par>)

sémantique (proc rechvar :

B511) En cas d'erreur : vers fin commune ;

B512) Sinon :

B5121) Pour <dim> = 0 et (<type> = "vl" ou = 'ind"):

erreur : mettre "er26" en <err> ;

B5122) Sinon : déplacer les caractéristiques,

0 en <ctrl> , blanc en identificateur;

B52) Manque un ident. : erreur: mettre "er27" en <err> ;

B6) Sinon : erreur : mettre "er28" en <err> ;

Commentaire Il est impossible de prévoir en ce point du travail, si un paramètre est ou non indicé. Cela est à la nature du langage traité. La procédure

qui est en chantier est destinée à être insérée dans une ou plusieurs procédures qui n'existent pas forcément encore. Ce n'est qu'au moment du déroulement et à l'instant de l'insertion qu'on peut savoir. Il faut ainsi se donner les moyens de faire la vérification. Où on en fait une propriété du compilateur de déroulement, où on rajoute en ce point du programme généré une insertion d'une procédure qui fait cette vérification.

---

par-fer

A) Quand on a <par> = "/" :

A1) Ferme un identificateur (<ident> = "/"):

syntaxe (mettre "0" en <ident> ,et "2" en <par>)

sémantique (proc rechvar :

A11) En cas d'erreur : vers fin commune ;

A12) Sinon :

A121) En cas où <type> = "ind" : erreur : mettre "er29" en <err> ;

A122) Sinon : vers fin commune ;

A2) Ferme un nombre (<nomb> = "/"):

A21) Quand <decl> = "2" :

syntaxe (mettre "2" en <par> ,et "0" en <nomb>)

sémantique (mettre <nb> en <dim> , proc charvar);

A22) Dans les instructions (<pro> = "2"):

syntaxe (mettre "2" en par> ,et "0" en <nomb>)

sémantique (:

A221) Quand on a (<cond> = "/" ou <aff> = "0"):

mettre "0" en <num> , proc genct);

A222) Ou bien (<cond> = "2" ou <aff> = "/"):

mettre 1 en <num> , proc genct);

A223) Ou bien (<aff> = "2"):

mettre 2 en <num>, proc genct);

B) Quand (<par> = "0"):

B1) Ferme une liste d'insertion (<inser> = "2"):

syntaxe (mettre "3" en <inser>)

sémantique (proc genct , <nos> + 3 + <num> en <no> , proc genepar ,

ajouter 1 en <num>);

B2) Sinon : erreur : mettre "er29" en <err> ;

C) Quand (<par> = 2):

C1) Et si suit une ")" , ferme une liste d'insertion :

syntaxe (mettre "3" en <inser> , et "0" en <par>)

sémantique (proc genparind , ajouter 1 en <npe>);

Commentaire Je vais détailler les procédures telles que : rechvar, genct genevar etc.

---

proc geninstl

Dans l'octet de référence q,1 mettre:

":=" X 2^9 + 4 X 2^3

Dans les deux octets suivants:

0

les deux suivants:

0

Ajouter 5 dans <vp>;

Déplacer <q> de 3 cases

Génère l'instruction qui met en place dans la case 0 de la configuration : le volume de la configuration. Au moment où on génère, ce volume est inconnu, on met 0

L_0 := 0

fin

---

proc adrvl

Dans l'octet de référence q mettre:

":=" X 2^9 + 4

Dans les deux octets suivants:

<no>

Dans les deux suivants:

0

Les deux suivants:

<nvvl>

Ajouter 7 dans <vp> ;

Déplacer <q> de 4 cases;

fin

Génère l'adresse de la variable locale ainsi calculée:

$\Gamma_{vli} := \Gamma_o - q$

vli vient de no

q vient de nvvl

---

proc genct

Dans le premier octet on met:

":=" X 2^9 + 4

Dans les deux autres:

<nos> + 3 + <num> = bi

Dans les deux suivants:

<nb> = c^{te}

On ajoute 5 à <vp>;

Déplacer <q> de 3 cases.

fin

genère

$\Gamma_{bi} := c^{te}$

---

proc genepar

Dans le premier octet on met:

":=" X 2^9 + 2^6

Dans les deux suivants:

0

Dans les deux suivants:

<npe> + 3

Dans les deux suivants:

<no>

On ajoute 7 à <vp> ;

Déplacer <g> de 4 cases;

fin

genère:

$\Gamma_{o,ni} := i$

---

proc genparind

Dans le premier octet:

":=" X 2^9 + 2^6

Dans les 3 suivants:

0 , <npe> + 3 , no2

Dans les 6 suivants:

":=" X 2^9 + 2^6 + 2^3 + 1,

et: 0 , <npe> + 3 , 0 , <npe> + 3 ,

<no> , 0

dans les 12 suivants.

Ajouter 20 en <vp> ;

déplacer <g> de 12 cases.

fin

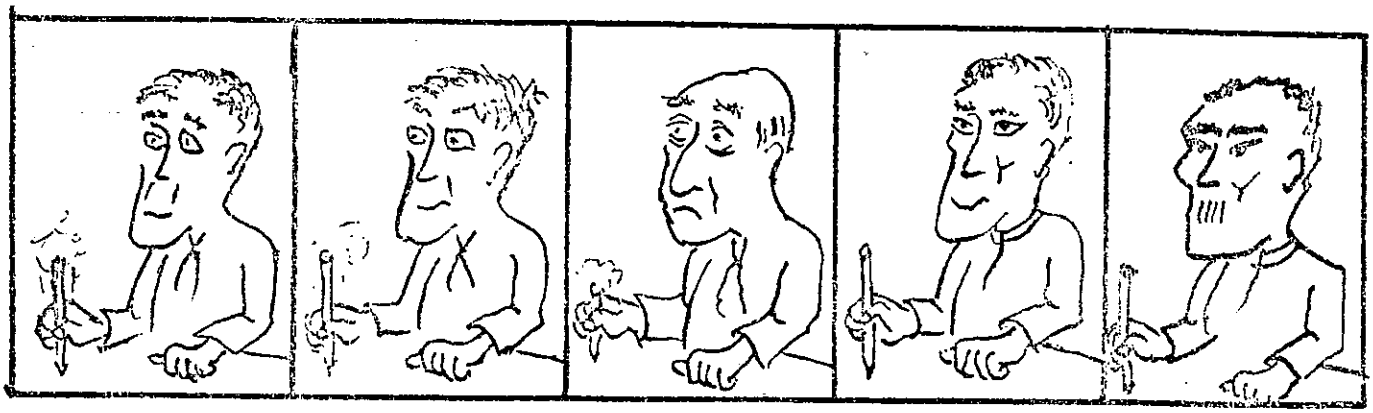
génère:

$\Gamma_{o,ni} := \Gamma_i$

$\Gamma_{o,ni} := \Gamma_{o,ni} + \Gamma_{j,o}$

à suivre...

VOUZZAVEDIBISAR



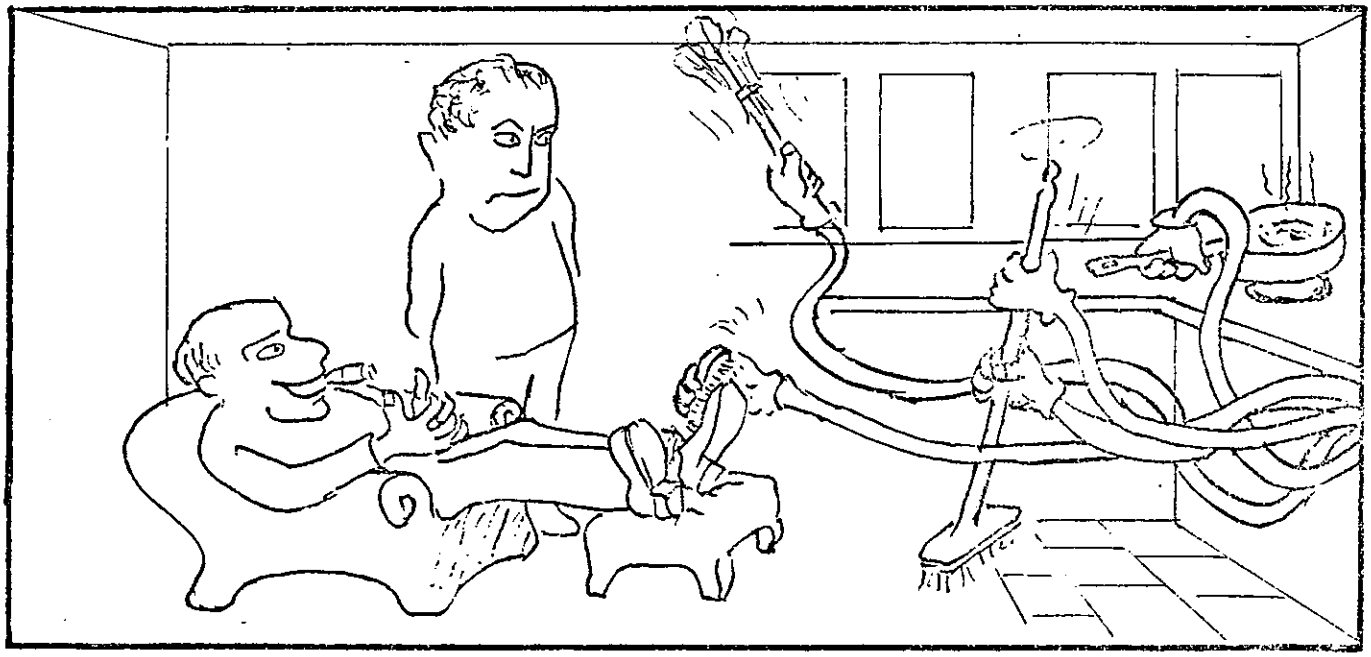
Jeune vrai-informaticien, plein d'allant.

Jeune faux-informaticien, plein d'allant.

Vieil informaticien fatigué.

Informaticien franco-japonais

Vrai futur informaticien



Et il me fait même la programmation...

Informaticien aux champs.