Programmation structurée

Autor(en): **Mange, D.**

Objekttyp: **Article**

Zeitschrift: **Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association Suisse des Electriciens, de l'Association des Entreprises électriques suisses**

Band (Jahr): **72 (1981)**

Heft 7

PDF erstellt am: **22.07.2024**

Persistenter Link: https://doi.org/10.5169/seals-905096

Programmation structurée

Par D. Mange

681.3.06;

Après un bref exposé des objectifs de la programmation structurée, on définit formellement les éléments qui la caractérisent: séquence (DO...), itération (WHILE... DO...) et test (IF... THEN... ELSE...). La structuration – c'est-à-dire la transformation d'un programme quelconque dans un programme équivalent, mais structuré – fait l'objet de deux méthodes: la première, intuitive, s'applique généralement à des parties de programme (structuration locale), tandis que la seconde, systématique, considère le programme dans sa totalité (structuration globale). Ces deux méthodes sont illustrées par l'exemple d'une montre digitale, dont le cahier des charges est décrit par l'algorithme horloger.

Nach einer kurzgefassten Darstellung der Ziele der strukturierten Programmierung werden die diese kennzeichnenden Elemente formell definiert: Sequenz (DO...), Iteration (WHILE... DO...) und Tests (IF... THEN... ELSE...). Zur Strukturierung, d.h. zur Umwandlung eines beliebigen Programmes in ein gleichwertiges, strukturiertes Programm, sind zwei Methoden möglich: Die intuitive Methode wird üblicherweise auf Teile eines Programms angewandt (lokale Strukturierung); die systematische Methode befasst sich mit dem ganzen Programm (globale Strukturierung). Diese beiden Methoden werden am Beispiel der Digitaluhr erläutert, deren Pflichtenheft durch den Uhren-Algorithmus beschrieben ist.

1. Introduction et définitions

1.1 Préambule

Le développement de circuits intégrés digitaux de plus en plus complexes (circuits pour microprocesseurs en tranches notamment) entraîne l'utilisation fréquente de systèmes logiques programmés. Il en découle le besoin de disposer de méthodes systématiques pour la réalisation du logiciel, c'est-à-dire pour la conception, le test et le dépannage des programmes ou microprogrammes. La programmation structurée constitue une réponse à ce besoin. Le but de cet article est de présenter brièvement les caractéristiques principales de la programmation structurée, puis de démontrer l'existence de méthodes permettant de transformer n'importe quel programme (non structuré) en un programme structuré équivalent.

1.2 Programmation structurée

La programmation structurée [1; 2; 3] est une méthode pour formuler et réaliser des algorithmes d'une façon systématique [4]. Les objectifs de la programmation structurée sont, dans l'ordre d'importance:

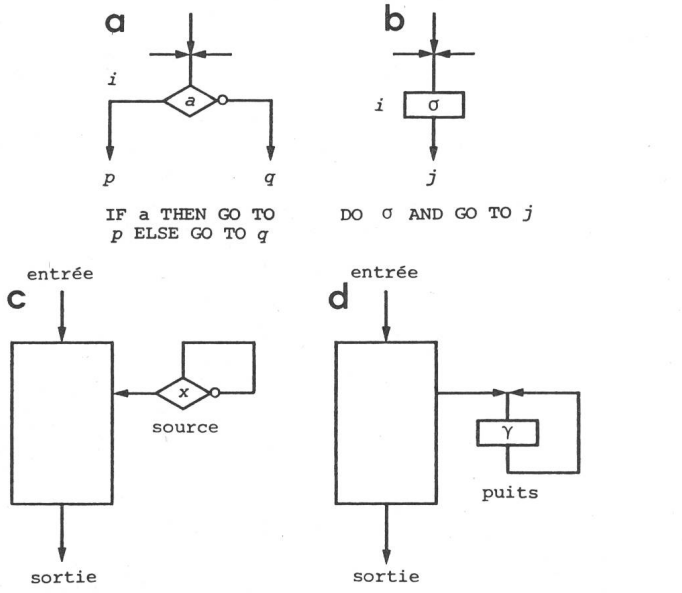

Fig. 1 Programme non structuré et ses éléments
a Instruction de test
b Instruction de sortie
c Programme avec une source
d Programme avec un puits

– la rédaction de programmes *corrects*; la programmation structurée rend plus faciles les techniques de validation.

– la rédaction de programmes *efficaces*; la programmation structurée facilite l'optimisation de certains paramètres, tels que la durée d'un cycle d'opérations ou la capacité de la mémoire (calcul systématique des arbres de décision binaire, par exemple).

– la rédaction de programmes *adaptables*; la programmation structurée permet de modifier une partie de programme sans affecter le tout *(modularité)*; la lecture ou la modification d'un programme par un tiers, la rédaction d'un programme par plusieurs auteurs sont également facilitées *(lisibilité)*.

Ces objectifs sont atteints à un certain prix, la redondance du logiciel. Le nombre plus élevé des instructions, ainsi que la complexité plus grande de celles-ci, entraînent une augmentation de taille de la mémoire (paramètre spatial); le nombre plus élevé d'instructions pour certaines actions allonge la durée d'un cycle (paramètre temporel). Cette redondance du logiciel peut donc diminuer l'efficacité du programme structuré et s'opposer ainsi aux objectifs originaux.

1.3 Programme non structuré

Un programme non structuré (ou, plus simplement, un *programme*) (fig. 4a p.ex.) est un assemblage quelconque de deux types d'éléments ou *instructions:*

– une instruction de *test* (IF...THEN GO TO... ELSE GO TO...), représentée par un losange (fig. 1a); chaque instruction de test, définie par un nombre ou adresse i et par une variable logique de test a, possède une borne d'entrée (reliée à une ou plusieurs instructions précédentes) et deux bornes de sortie conduisant chacune à une seule instruction suivante d'adresse p ($a = 1$) ou d'adresse q ($a = 0$); la borne de sortie de la variable complémentée ($a = 0$) est repérée par un rond accolé au losange.

– une instruction de *sortie* (DO... AND GO TO...), représentée par un rectangle (fig. 1b); chaque instruction de sortie, définie par une adresse i et par une action σ, possède une borne d'entrée (reliée à une ou plusieurs instructions précédentes) et une borne de sortie reliée à une seule instruction suivante d'adresse j.

Un organigramme non structuré (ou *organigramme*) est une représentation graphique d'un programme non structuré (fig. 4a, p.ex.), obtenu par l'assemblage des instructions données par les figures 1a et 1b.

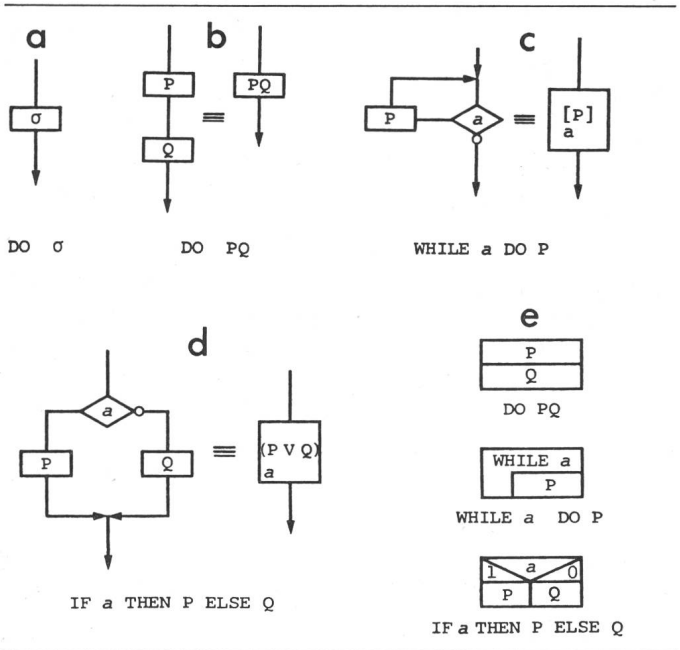

Fig. 2 Programme structuré
a Instruction de sortie
b Séquence ou composition de P et Q
c Itération ou répétition conditionnelle de P
d Test de P ou Q
e Structogrammes

1.4 Programme propre

Un programme propre (fig. 4b p.ex.) est un programme [5]

– qui comporte une entrée et une sortie;

– dont toutes les instructions sont accessibles à partir de l'entrée; il n'existe donc aucune *source* (fig. 1c p.ex.);

– dont toutes les instructions conduisent à la sortie; il n'existe donc aucun *puits* (fig. 1d p.ex.).

1.5 Programme structuré [6; 7]

(1) L'instruction de sortie (DO...), représentée par un rectangle (fig. 2a), définie par une action σ, possédant une borne d'entrée (reliée à une seule instruction précédente) et une borne de sortie (reliée à une seule instruction suivante), est un programme structuré. Si P et Q sont des programmes structurés, alors

(2) la *séquence* ou *composition* de P et Q (DO PQ), notée PQ, est un programme structuré (fig. 2b); les programmes structurés P et Q sont les *éléments* de la séquence PQ;

(3) l'*itération* ou *répétition conditionnelle* de P (WHILE *a* DO P), notée $\underset{a}{[P]}$, est un programme structuré (fig. 2c); un tel programme possède une *boucle de rétroaction*;

(4) le test de P ou Q (IF *a* THEN P ELSE Q), notée $\underset{a}{(PVQ)}$, est un programme structuré (fig. 2d).

(5) Il n'y a pas d'autres programmes structurés que ceux obtenus en appliquant les définitions (1) à (4) un nombre fini de fois.

1.6 Organigramme structuré et structogramme

Un organigramme structuré (fig. 7a p.ex.) est une représentation graphique d'un programme structuré, obtenu par l'assemblage des éléments donnés par les figures 2a, 2b, 2c et 2d. Lorsque ces mêmes éléments sont représentés sous la forme particulière de la figure 2e, leur assemblage porte le nom de structogramme (fig. 7b, p.ex.) [8].

1.7 Equation d'un programme structuré

La concaténation des expressions algébriques P, Q, PQ, $\underset{a}{(PVQ)}$, $\underset{a}{[P]}$ définissant les éléments d'un programme structuré, constitue l'équation de ce programme. A titre d'exemple, l'équation du programme x (fig. 6b) s'écrirait

$$x = \varrho_1 \underset{s}{[} \underset{T}{(} T_0 \underset{i}{(} \varrho_0 \vee T_1 \varrho_1) \vee \varrho_0)]$$

L'étude de ces équations fait notamment l'objet des références [7; 9; 10].

1.8 Commentaire

Tous les programmes structurés ont donc une seule entrée et une seule sortie; la composition quelconque de plusieurs programmes structurés produit toujours un programme structuré (exemple: fig. 7a). Contrairement aux programmes non structurés, il n'est pas nécessaire de connaître les adresses futures (GO TO...) pour décrire un programme structuré: la composition des éléments de ce programme suffit [11].

1.9 Structuration

La structuration d'un programme propre, non structuré, est la transformation de celui-ci dans un programme équivalent, mais structuré. La structuration est *locale* si la transformation n'affecte qu'une partie du programme non structuré, elle est *globale* dans le cas contraire.

1.10 Conclusion

L'étude des méthodes systématiques de structuration, qui fait l'objet de cette publication, est entreprise dans le double but

– de démontrer que tout programme non structuré (et propre) est réalisable par un programme structuré, et de prouver ainsi l'universalité de la programmation structurée;

– de réaliser un programme structuré à partir d'un cahier des charges représenté par un programme propre, non structuré.

2. Structuration locale

2.1 Méthode

Elle consiste en l'application successive et, éventuellement, répétée des trois règles qui suivent; ces règles découlent directement des définitions de la programmation structurée.

2.2 Règle de structuration n° 1: composition ou décomposition

Soit deux programmes propres p et q assemblés en séquence selon la figure 3a. Le programme résultant τ est aussi propre. Selon le contexte du problème (et selon l'application possible des deux autres règles), on peut effectuer la structuration de chacun des deux programmes p et q (décomposition) ou, au contraire, la structuration du seul programme résultant τ (composition).

2.3 Règle de structuration n° 2: coupure des boucles

Soit une boucle de rétroaction appartenant à un organigramme non structuré et ne comportant qu'une seule borne d'entrée (fig. 3b et 3c). Le nombre des bornes de sortie de cette boucle est la *sortance* (en anglais: fan out).

Toute boucle dont la sortance est égale à *un* peut être réalisée par un programme structuré comportant un élément WHILE... DO... (fig. 3b).

Définition des notations de l'algorithme horloger **Table I**

Indice i (décimal)	Indice i (binaire) $i_2\ i_1\ i_0$	Variable $T(i)$	Constante $G(i)$
0	0 0 0	$T(0)$ = unité de seconde	$G(0) = 10$
1	0 0 1	$T(1)$ = dizaine de secondes	$G(1) = 6$
2	0 1 0	$T(2)$ = unité de minute	$G(2) = 10$
3	0 1 1	$T(3)$ = dizaine de minutes	$G(3) = 6$
4	1 0 0	$T(4)$ = unité d'heure	$G(4)$ { = 10 pour 00...19 heures = 4 pour 20...23 heures
5	1 0 1	$T(5)$ = dizaine d'heures	$G(5) = 3$
$6 = i_{MAX}$	1 1 0	–	–
7	1 1 1	–	–

2.4 Règle de structuration n° 3: abaissement des boucles [12]

Soit une boucle dont la sortance est supérieure à un (fig. 3c): dans un tel cas, il n'est pas possible d'obtenir directement un programme structuré, à moins de recourir à la méthode de structuration globale exposée plus loin.

Cependant, dans le cas où le premier élément de la boucle est un programme propre p, il est possible, par duplication de ce programme, d'obtenir un *abaissement de la boucle* mis en évidence par la figure 3c.

2.5 Cahier des charges: montre digitale

Une montre digitale, excitée par un signal d'horloge H, doit compter et afficher les grandeurs suivantes:

- les secondes (de 00 à 59);
- les minutes (de 00 à 59);
- les heures (de 00 à 23).

Le signal d'horloge H est une variable logique dont la période est égale à une seconde; l'incrémentation des secondes doit être effectuée à chaque montée de H ($H = 0 \rightarrow 1$).

2.6 Algorithme horloger

Il existe un algorithme universel pour la mesure du temps, généralement appelé algorithme horloger [13; 14; 15]. En adoptant les notations de la table I, l'algorithme horloger, représenté par l'organigramme de la figure 4a, réalise le cahier des charges de la montre digitale de la façon suivante:

– l'indice i indique la variable calculée; de 0 à 5, on distinguera tour à tour les secondes (unités, puis dizaines), les minutes (unités, puis dizaines) et les heures (unités, puis dizaines);

– la grandeur $T(i)$ est la variable calculée, dont la valeur décimale est comprise entre 0 et 9;

– la constante $G(i)$ constitue la limite supérieure de la valeur de la variable $T(i)$.

En partant alors du premier test de H («START» sur la fig. 4a), le programme détecte la montée du signal H (second test de H) et force l'indice i à la valeur 0 ($i \leftarrow 0$: comptage des unités de seconde); la valeur présente des secondes $T(0)$ est alors incrémentée ($T(0) \leftarrow T(0) + 1$); si cette nouvelle valeur n'est pas égale à la limite $G(0) = 10$, alors l'incrémentation des secondes se poursuit pour chaque montée de H; sinon, les unités de secondes sont mises à zéro ($T(0) \leftarrow 0$) et l'indice i est incrémenté ($i \leftarrow i + 1 = 1$): le programme s'apprête à compter les dizaines de secondes ($T(1) \leftarrow T(1) + 1$). Le processus se poursuit jusqu'à la détection de la valeur limite de l'indice ($i_{max} = 6$), qui coïncide avec la remise à zéro complète de la montre.

2.7 Programme propre

Le programme de la figure 4a n'est pas propre: il ne comporte ni entrée, ni sortie. L'introduction d'une variable logique auxiliaire E (Enable) permet d'obtenir aisément un organigramme propre (fig. 4b, PP: programme principal); dans celui-ci on a, par ailleurs, simplifié le libellé de certaines instructions.

On remarque que l'organigramme propre est obtenu par la coupure d'une boucle de l'organigramme original (fig. 4a) et l'insertion d'un test sur E; cette insertion est telle que ce test est effectué chaque seconde. Une autre coupure possible entraînerait le test sur E chaque fois que $i = i_{MAX}$, c'est-à-dire chaque jour.

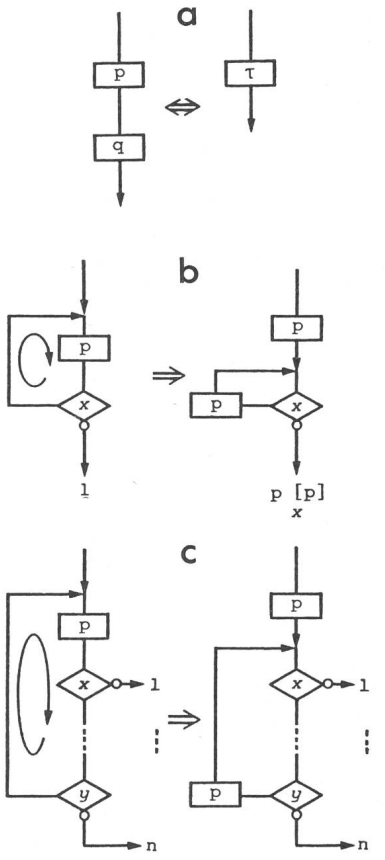

Fig. 3 Règles de structuration
a Règle n° 1: décomposition ou composition
b Règle n° 2: coupure de boucle
c Règle n° 3: abaissement de boucle

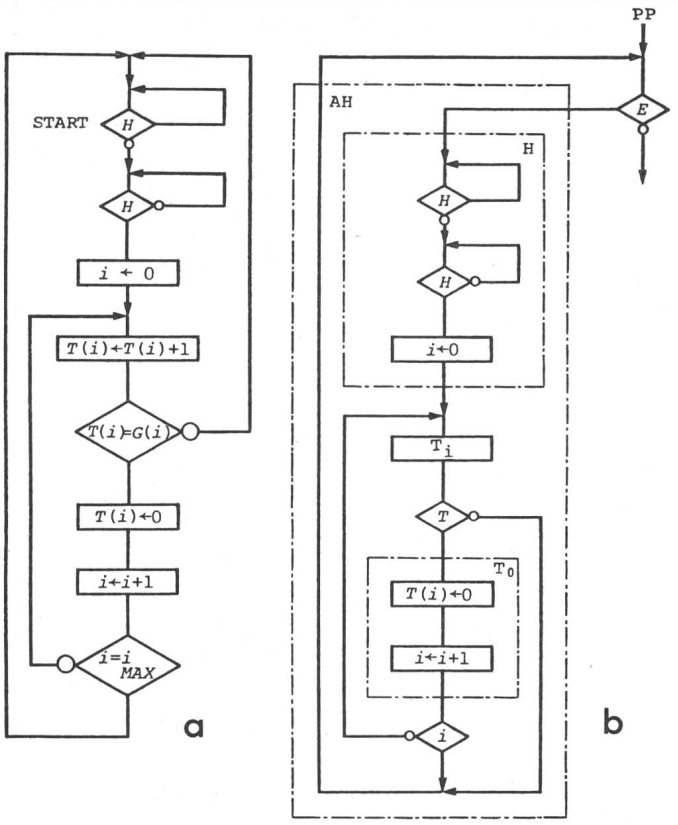

Fig. 4a, 4b L'algorithme horloger
a Organigramme original
b Organigramme propre; E = Enable

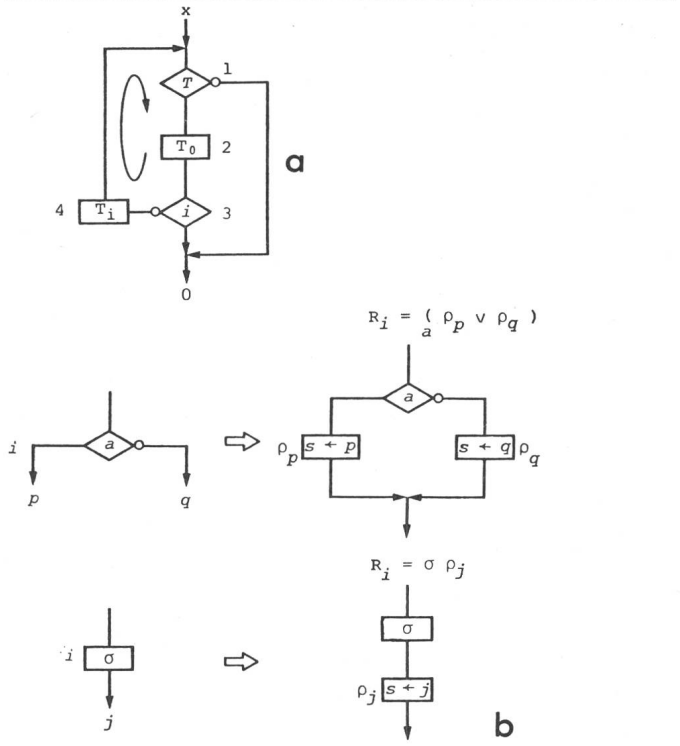

Fig. 5 Structuration globale
a Organigramme original
b Transformations générales

2.8 Exemple de structuration locale

L'application des règles de structuration est illustrée de la façon suivante:

– La règle nº 1 transforme l'organigramme de la figure 4b dans les deux organigrammes de la figure 4c; les nouveaux programmes propres H, T_0 et AH sont obtenus par composition; le programme principal (PP) est structuré, de même que les programmes H et T_0: ils sont distingués par un astérisque (*).

– La règle nº 3 transforme l'organigramme AH de la figure 4c dans celui de la figure 4d par abaissement d'une boucle.

– La règle nº 1 transforme l'organigramme AH de la figure 4d dans celui de la figure 4e par composition (nouveaux programmes propres h et x; h est structuré).

A l'exception du programme propre x (fig. 4d), tous les programmes obtenus sont structurés. L'application des trois règles de structuration au programme x n'est plus possible: il faut donc recourir à la méthode de structuration globale.

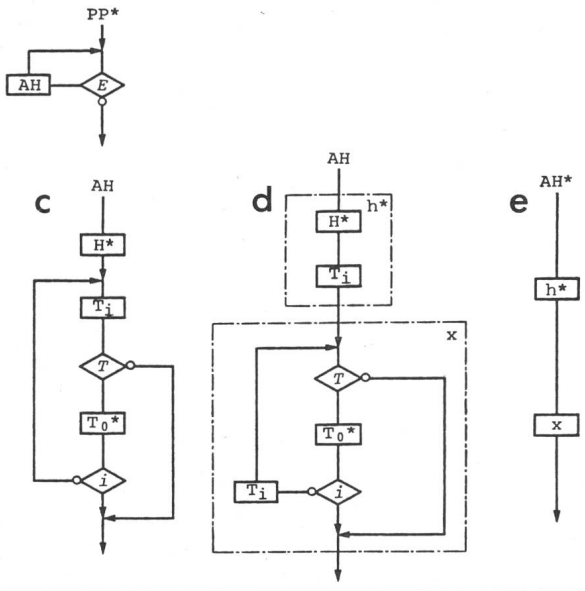

Fig. 4c, 4d, 4e Application des règles de structuration locale
c Application de la règle nº 1
d Application de la règle nº 3
e Application de la règle nº 1

2.9 Conclusion

La méthode de structuration locale est facile à mettre en œuvre; elle n'est pas systématique, et le résultat peut dépendre de l'ordre dans lequel les trois règles ont été appliquées.

Par ailleurs, l'existence d'une ou plusieurs boucles avec une sortance supérieure à un, ne permet qu'une structuration partielle du programme original; seule la méthode de structuration globale peut achever la transformation.

3. Structuration globale

3.1 Méthode [5; 16]

Celle-ci est illustrée par la transformation du programme propre x (fig. 5a), qui constitue la partie non structurée de l'algorithme horloger. Les étapes de la méthode sont les suivantes:

(1) Toutes les instructions du programme original sont munies d'une adresse i arbitraire (i = 1, 2, 3, 4 dans l'exemple traité); l'adresse i = 1 est réservée à la première instruction

rencontrée à l'entrée du programme; l'adresse $i = 0$ est affectée à la sortie du programme.

(2) Chaque instruction du programme original ($i = 1, 2, 3, 4$) est remplacée par un programme structuré selon la transformation de la figure 5b. On introduit donc une variable auxiliaire s, qui est multivaluée ($0 \leqq s \leqq 4$).

(3) L'organigramme structuré recherché se compose (fig. 6a)

– d'une *partie universelle,* identique pour chaque problème de structuration, à l'exception du nombre des tests qui est égal au nombre d'instructions du programme original (quatre dans l'exemple traité);

– d'une *partie spécialisée* qui regroupe tous les programmes structurés produits dans l'étape précédente ($R_1 ... R_4$).

3.2 Organigramme canonique structuré

L'organigramme obtenu par la méthode de structuration globale est appelé organigramme canonique structuré: il est unique pour un organigramme (non structuré) donné. On vérifie aisément que l'organigramme canonique structuré

– représente un programme structuré;

– comporte un seul élément WHILE... DO... et une unique boucle de rétroaction;

– exige n tests de la variable auxiliaire s [s est $(n + 1) -$ valuée], si n est le nombre d'instructions de l'organigramme original.

3.3 Organigramme structuré simplifié

Il est possible de simplifier l'organigramme canonique structuré, c'est-à-dire d'en diminuer le nombre d'éléments, en procédant comme suit: Chaque élément ϱ_j (DO $s \leftarrow j$) est remplacé par le programme structuré R_j, à l'exception de ϱ_0 (DO $s \leftarrow 0$) qui reste inchangé.

Dans l'exemple traité (fig. 6a), on peut procéder aux remplacements successifs suivants:

– dans le programme R_1, ϱ_2 est remplacé par R_2:

$$R_1 = \underset{T}{(\varrho_2 \vee \varrho_0)} = \underset{T}{(T_0\ \varrho_3 \vee \varrho_0)};$$

– dans le programme R_2, ϱ_3 est remplacé par R_3:

$$R_1 = \underset{T\ \ i}{(T_0\ (\varrho_0 \vee \varrho_4)} \vee \varrho_0);$$

– dans le programme R_3, ϱ_4 est remplacé par R_4 (fig. 6b):

$$R_1 = \underset{T\ \ i}{(T_0\ (\varrho_0 \vee T_i\ \varrho_1)} \vee \varrho_0).$$

Dans le programme R_4 (fig. 6a), il apparaît ϱ_1 ($s \leftarrow 1$): on obtient le cas particulier où l'élément ϱ_j renvoie au programme R_j auquel il appartient; cet appel récursif empêche alors le remplacement énoncé par la règle de simplification, et le programme structuré final s'exprime enfin par l'équation:

$$x = \underset{s}{\varrho_1\ [R_1]} = \underset{s\ T\ \ i}{\varrho_1\ [\ (T_0\ (\varrho_0 \vee T_i\ \varrho_1)} \vee \varrho_0)\ ]$$

L'organigramme correspondant (fig. 6b) est un organigramme structuré simplifié; pour un organigramme canonique donné, il existe un grand nombre d'organigrammes structurés simplifiés, dont la forme finale dépend de l'ordre dans lequel les remplacements successifs ont été effectués.

On constate, dans l'exemple traité (fig. 6b), que l'organigramme structuré simplifié

– représente un programme structuré;

– comporte un nombre d'éléments inférieur à celui de l'organigramme canonique (9 au lieu de 15);

– comporte un seul élément WHILE... DO... et une unique boucle de rétroaction;

– exige k tests de la variable auxiliaire s [s est $(k + 1)$-valuée], si k est le nombre de boucles essentielles de l'organigramme original, non structuré (dans le cas général, on a $k < n$, où n est le nombre d'instructions de l'organigramme original).

Dans l'exemple traité, il existe une seule boucle essentielle dans l'organigramme original, non structuré (fig. 5a); il existe par conséquent, dans l'organigramme structuré simplifié, un unique test sur la variable auxiliaire s, qui est binaire (2-valuée) (fig. 6b).

3.4 Programme structuré final

L'assemblage des programmes structurés x (fig. 6b), h (fig. 4d), H (fig. 4b) et PP (fig. 4c) produit le programme final qui est complètement structuré (fig. 7a). Dans ce programme, on a restauré les libellés originaux (selon fig. 4a) et, pour plus de lisibilité, on a décomposé le programme structuré x en un élément WHILE... DO... et un programme structuré y.

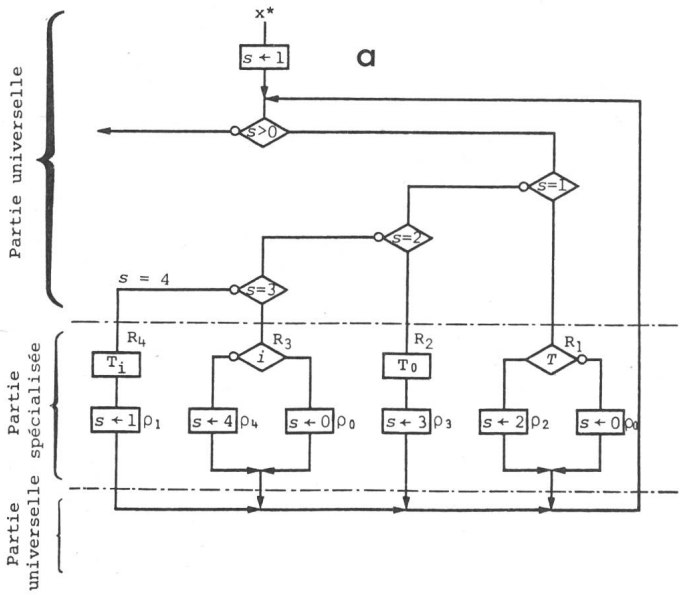

Fig. 6a Organigramme canonique structuré

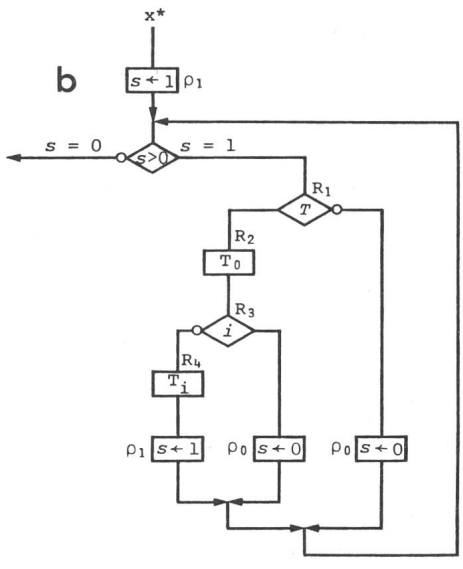

Fig. 6b Organigramme structuré simplifié

Le structogramme équivalent à l'organigramme structuré (fig. 7a) est donné à la figure 7b, dans laquelle le mnémonique NOP (no operation) représente une opération neutre (c'est-à-dire une instruction de sortie sans effet).

La figure 7c donne enfin le programme décrit par les mnémoniques DO..., WHILE... DO..., IF... THEN... ELSE..., définis au paragraphe 1.5.

3.5 Conclusion

La structuration globale du programme propre original (fig. 4b) produirait un organigramme canonique structuré à neuf tests (avec une variable *s* 10-valuée) et un seul élément WHILE... DO... (avec une seule boucle de rétroaction).

La structuration locale du même programme a diminué la complexité de la structuration globale, appliquée à la seule partie x du programme original. Le programme structuré final comporte un seul test (avec une variable *s* binaire) et quatre éléments WHILE... DO..., donc quatre boucles de rétroaction.

L'intérêt principal de la méthode de structuration locale, outre sa simplicité, réside dans la réduction de complexité qu'elle entraîne pour l'application de la structuration globale.

4. Conclusion générale

4.1 Méthodes de structuration

Dans la pratique, on recommande de construire un programme structuré directement à partir de son cahier des charges. Les méthodes de structuration semblent cependant indispensables:

– pour démontrer l'universalité de la programmation structurée (besoin théorique);

– pour réaliser un cahier des charges, lorsque celui-ci est représenté par un organigramme non structuré (besoin pratique).

La structuration locale du programme original, suivie d'une éventuelle structuration globale pour les éléments restant non structurés (c'est-à-dire les boucles dont la sortance est supérieure à un), constitue une approche possible du problème posé.

4.2 Méthode de synthèse descendante

Tout processeur admet une décomposition en deux parties, généralement appelées unité de commande (ou de contrôle) et

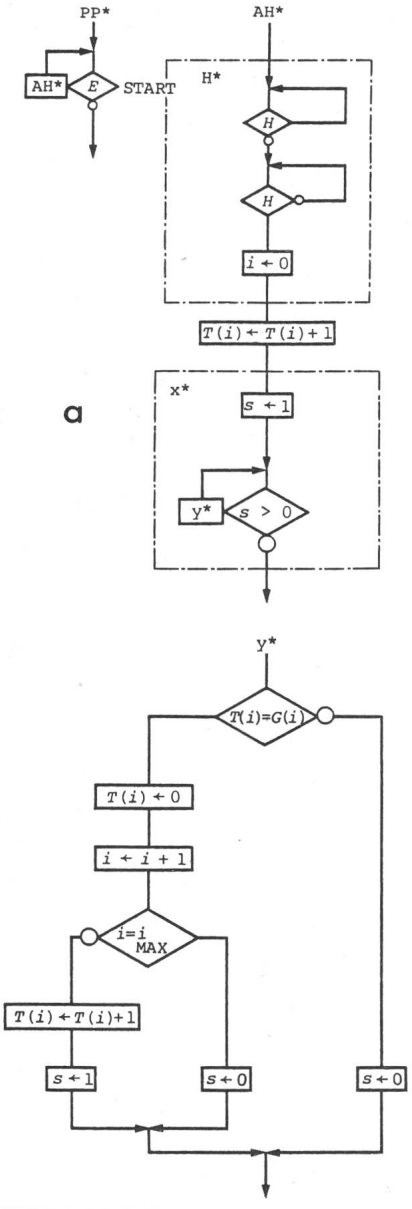

Fig. 7a Organigramme structuré final de l'algorithme horloger

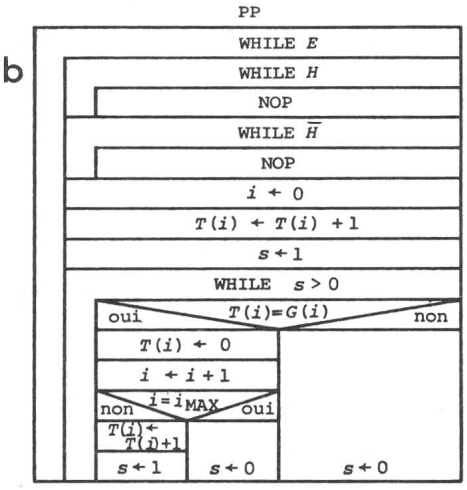

Fig. 7b Structogramme final

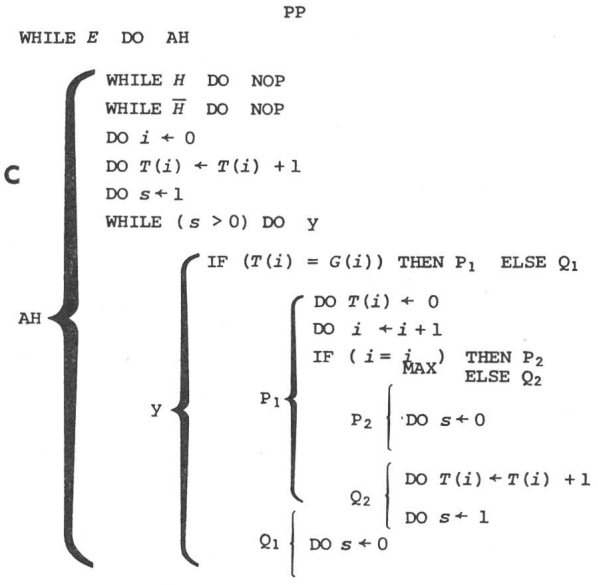

Fig. 7c Programme mnémonique final

unité de traitement (ou unité opérative). Examinons l'exécution par un tel processeur d'une instruction de l'algorithme horloger (fig. 7a), par exemple celle qui réalise l'incrémentation de l'indice i:

DO $i \leftarrow i + 1$.

Si l'unité de traitement est assez puissante, c'est-à-dire si elle dispose d'un circuit arithmétique permettant l'incrémentation d'un nombre i, alors le programme de l'unité de commande se borne à envoyer un ordre simple. Si, par contre, l'unité de traitement se résume à l'existence de registres de sortie, comme c'est le cas dans les machines de décision binaire décrites dans [11; 17; 18], alors l'action $i \leftarrow i + 1$ doit être réalisée par un assemblage d'instructions plus simples, se ramenant en général à des tests d'une variable logique et des transferts, dans un registre, d'un état logique. Le raffinement successif d'une action complexe (par exemple $i \leftarrow i + 1$) en un programme structuré ne comportant que des actions simples (IF a THEN P ELSE Q, DO $R_j \leftarrow 1001$, par exemple), constitue la synthèse descendante de ce programme (en anglais: top down methodology).

La synthèse descendante d'un programme structuré fait l'objet de travaux de recherche du Laboratoire de systèmes logiques de l'EPFL. La réalisation d'unités de commande spécialisées dans l'exécution de microprogrammes structurés a déjà fait l'objet d'études antérieures [11].

Bibliographie

[1] *Y. Tabourier, A. Rochfeld* et *C. Frank:* La programmation structurée en informatique. Paris, Editions d'organisation, 1975.

[2] *W. Findlay* and *D.A. Watt:* Pascal, an introduction to methodical programming. Potomac/Maryland, Computer Science Press, 1978.

[3] *B. Meyer* et *C. Baudoin:* Méthodes de programmation. Collection de la direction des études et recherches de l'Electricité de France. Paris, Eyrolles, 1978.

[4] *N. Wirth:* Introduction à la programmation systématique. Paris, Masson, 1977.

[5] *R.C. Linger, H.D. Mills* and *B.I. Witt:* Structured programming, theory and practice. Reading/Massachusetts, Addison-Wesley, 1979.

[6] *C. Böhm* and *G. Jacopini:* Flow diagrams, Turing machines and languages with only two formation rules. Communications of the Association for Computing Machinery 9(1966)5, p. 366...371.

[7] *M. Davio:* Hardware implementation of algorithmic computations. MBLE Research Laboratory, Report R 333. Brussels, Manufacture Belge des Lampes et de Matériel Electronique, 1976.

[8] *H. Bühler:* Cours d'automatisation de processus. Vol. Ib. Lausanne, Ecole Polytechnique Fédérale de Lausanne, 1979.

[9] *T. Ito:* A theory of formal microprograms. Sigmicro Newsletters 4(1973)1, p. 5...17.

[10] *V.M. Glushkov:* Automata theory and formal microprogram transformations. Cybernetics 1(1965)5, p. 1...9.

[11] *D. Mange:* Microprogrammation structurée. Le Nouvel Automatisme 25(1980) 13, p. 45...54.

[12] *R. Kosaraju:* Analysis of structured programs. Journal of Computer and System Sciences 9(1974)3, p. 232...252.

[13] *C. Piguet, J.-F. Perotto* et *J.-J. Monbaron:* Conception d'un microprocesseur horloger. Proceedings du Congrès International de Chronométrie, Genève, 10(1979)3, p. 271...278.

[14] *J.-J. Monbaron* e.a.: Conception d'un microprocesseur horloger. Journées d'Electronique 1979 de l'Ecole Polytechnique Fédérale de Lausanne, p. 207 à 216.

[15] *J.-J. Monbaron,* et *N. Peguiron:* Conception de microprocesseurs spécialisés: exemple du microprocesseur horloger. Bull. ASE/UCS 71(1980)11, p. 558...562.

[16] *D. Harel:* On folk theorems. Communications of the Association for Computing Machinery 23(1980)7, p. 379...389.

[17] *D. Mange:* Arbres de décision pour systèmes logiques câblés ou programmés. Bull. ASE/UCS 69(1978)22, p. 1238...1243.

[18] *D. Mange:* Compteurs microprogrammés. Bull. ASE/UCS 70(1979)19, p. 1087 à 1095.

Adresse de l'auteur

Daniel Mange, Professeur EPFL, Laboratoire de systèmes logiques,
16, chemin de Bellerive, 1007 Lausanne.