TECHNION - Israel Institute of Technology
Computer Science Department

ON PARALLEL PROGRAMMING PRIMITIVES

by

M. Yoeli and A. Ginzburg*

Technical Report #351

January 1985

* Computer Science Dept., Technion and Everyman's University, Tel Aviv, Israel.

ON PARALLEL PROGRAMMING PRIMITIVES

M. Yoeli
Department of Computer Science
Technion - Israel Institute of Technology
Haifa, Israel

and

A. Ginzburg
Department of Computer Science
Technion - Israel Institute of Technology
and
Everyman's University
Tel Aviv, Israel

ON PARALLEL PROGRAMMING PRIMITIVES

M. Yoeli and A. Ginzburg

ABSTRACT

Structured programming is now widely recognized as an essential tool for the design of correct, easily understood programs. A key issue in structured programming is the suitable choice of the set of control structures to be used. As far as structured sequential programming is concerned, important theoretical results are available on the "relative power" of various classes of control structures. This paper discusses this "relative power" issue for classes of parallel control structures. It establishes a mathematically precise framework in which all the relevant results are presented.

Only a restricted class of parallel programs is considered in this paper. These programs can be represented by one-in, one-out cycle-free structures containing basic action modules and two types of control modules: 2-way forks and 2-way joins. In particular, we demonstrate the limitations of any finite set of control primitives: there always exists a parallel program not structurable by means of the given set of primitives.

1. INTRODUCTION

Structured programming has become an important methodology for the design of correct, easily understood computer programs [DA-DI-HO]. The arguments in favor of a structured approach to sequential programming evidently also apply to parallel programming and parallel processing. An important aspect of structured programming is the appropriate selection of control primitives. This paper is a contribution towards a formal theory of parallel control primitives. Such a theory is also applicable to the structured design of asynchronous control networks (cf. [BRU-ALT], [HE-YO], [YOE], [CO-MA], [YO-GI]).

2. TASK FLOW CHARTS

In this section we discuss in an informal way the problems we shall be concerned with in the sequel. All the notions mentioned in this section will be made precise later on.

Let us consider a system dedicated to some overall objective. Such a task of a system can usually be decomposed into several subtasks; some of which may be executable simultaneously (<u>in parallel</u>). A <u>task flow chart</u> [BR-YO] indicates the (partial) order, in which the subtasks have to be performed. We assume that the overall system is initiated by a START-command, and that it issues a DONE-signal upon completion of its overall objective. A task flow chart for some hypothetical system SYS1 is shown in Fig. 2.1. A directed path from TA_i to TA_j indicates that task TA_j may be started only after the completion of taks TA_i.

Let us denote by [TA1||TA2] a task consisting of two subtasks TA1 and TA2 which <u>may be</u> executed in parallel. Similarly (TA1;TA2) will

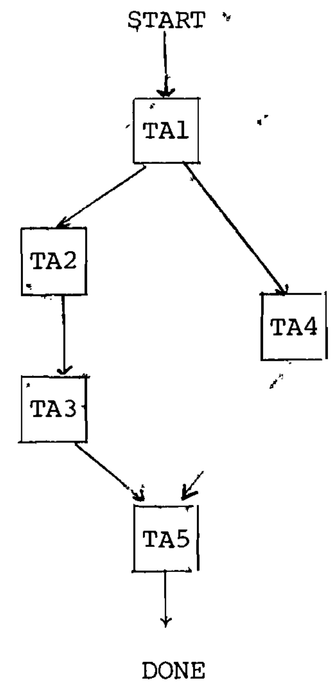


Figure 2.1 - Task flow chart for hypothetical system SYS1.

denote a task composed of subtasks TA1 and TA2, to be executed sequentially (TA1 first). These or equivalent notions appear in many modern programming languages (cf. [WE-SM]) as primitive constructs. Evidently, the overall task TA of SYS1 may be represented in a structured form as follows:

$$TA = (TA1;[(TA2;TA3) \parallel TA4];TA5). \tag{2.1}$$

Consider how the overall task represented by the task flow chart of Fig. 2.2. It will be shown later (see Section 7) that this task cannot be "structured" in a form similar to (2.1) above, without introducing additional constraints. Thus, the above two control primitives ($\parallel$ and ;) are not powerful enough for the structuring of arbitrary composite tasks.

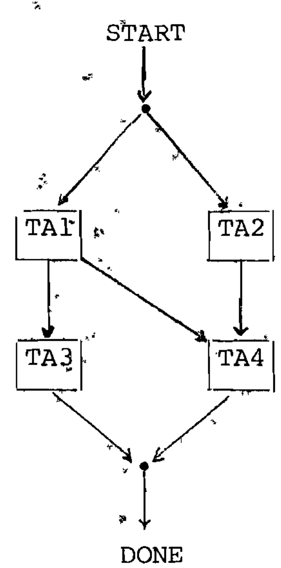


Figure 2.2 - A task flow chart not structurable by '||' and ';'

We are thus confronted with the problem of suitably selecting additional control primitives and of determining the increased structuring power obtained. Although this problem is, no doubt, of interest, it seems that it has not received suitable attention in the literature.

On the other hand, the corresponding problem related to structured sequential programming has been investigated extensively (cf. [LE-MA]). As to structured parallel processing, various sets of control primitives have been proposed [BRU-ALT], [KEL], [YOE], [VAL], [WEI], without, however, investigating the limits of their structuring power.

In Sections 4-8 we develop a formal theory which will enable us to deal with the above problem of structured parallel processing in a precise way. In our formal theory the concept of synchronization graph plays an important role. This concept is introduced in the following section.

3. SYNCHRONIZATION GRAPHS - INFORMAL INTRODUCTION

The control part of SYS1 (see Fig. 2.1) may be implemented as shown by the control (or synchronization) graph of Fig. 3.1. A node labeled TA_i represents the task module executing task TA_i. The inedge (outedge) of this node represents the control input (output) of the corresponding module. Any task module operates asynchronously: it starts its operation as soon as a signal (start command) arrives on its control input. Upon completion of the operation, the module issues a (completion) signal on its control output.

Nodes others than those labeled TA_i represent control modules.

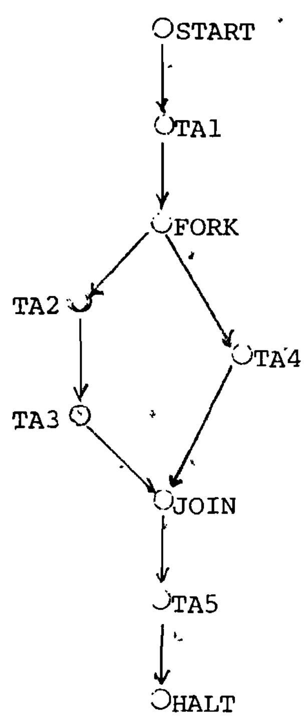


Figure 3.1 - Synchronization graph (control graph) for SYS1.

A(2-way) FORK is a one-input, two-output control module, which issues signals, one on each output, after having received a signal on its input.

A (2-way) JOIN is a two-input, one-output control module, which issues a signal on its output, after having received signals on both its inputs.

The START-module initiates the overall operation of the system by issuing a signal on its output. The overall operation is completed as soon as a signal arrives on the input of the HALT-module.

4. SYNCHRONIZATION GRAPHS - FORMAL DEFINITION

This section, which contains the formal definition of synchronization graph, is a modified version of Section 2 of [GI-YO].

Definition: A synchronization graph (S-graph) is a finite, directed graph Γ, the nodes of which are partitioned into 5 types as shown in Fig. 4.1; furthermore, Γ satisfies the following conditions:

a) Multiple edges are not admitted.

b) Γ has exactly one START node S and exactly one HALT node H.

c) Every node v is reachable from S, i.e., there exists a (directed) path from S to v.

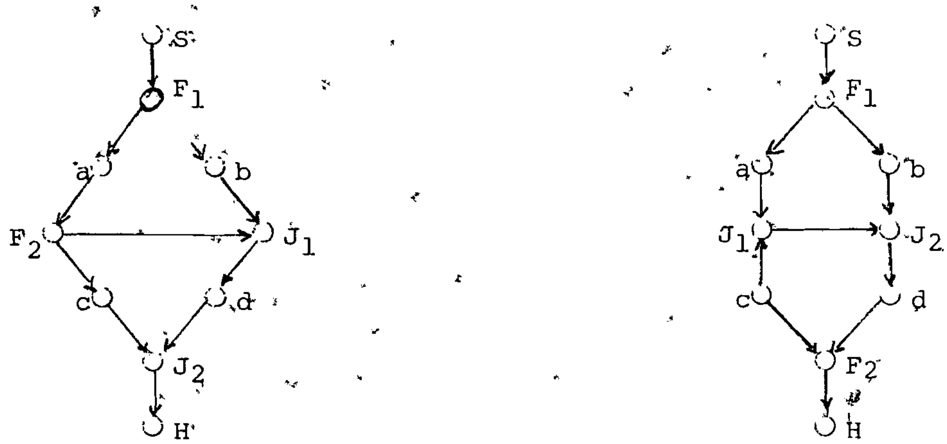
d) The node H is reachable from every node v.

NODE TYPE	INDEGREE	OUTDEGREE
START	0	1
HALT	1	0
FORK	1	2
JOIN	2	1
OPERATION	1	1

Figure 4.1 - Node types of S-graphs

Evidently, an S-graph cannot have self-loops (i.e. cycles of length 1). Examples of S-graphs are shown in Fig. 3.1 and Fig. 4.2.

<u>Definition</u> Let Γ be an S-graph. A <u>marking</u> m of Γ is a function $m: E \to \omega$, where E is the edge-set of Γ and ω is the set of nonnegative integers. A <u>marked S-graph</u> is an ordered pair (Γ, m) where Γ is an S-graph and m is a marking of Γ.



(a) (b)

<u>Figure 4.2</u> - Examples of S-graphs
(a) S-graph Γ_1
(b) S-graph Γ_2

Let e be an edge of the marked S-graph (Γ, m). We refer to the integer $m(e)$ as the number of <u>tokens</u> on e. If $m(e) > 0$, we say that e is <u>marked</u>. In the graphical representation of marked S-graphs, tokens are indicated by dots (•). Fig. 4.3 shows examples of marked S-graphs.

(a) (b)

<u>Figure 4.3</u> - Examples of marked S-graphs

<u>Definition</u> Let (Γ, m) be a marked S-graph. A node of type OPERATION or FORK is <u>enabled</u> iff its inedge is marked. A JOIN node is <u>enabled</u> iff both its inedges are marked.

A node which is enabled may <u>fire</u>. The firing rules, illustrated in Fig. 4.4, are as follows:

<u>Definition</u>

(a) The <u>firing</u> of a FORK node decreases the marking of its inedge by 1 and increases the marking of both its outedges by 1.

(b) The <u>firing</u> of a JOIN node decreases the markings of both its inedges by 1, and increases the marking of its outedge by 1.

(c) The <u>firing</u> of an OPERATION node decreases the marking of its inedge by 1, and increases the marking of its outedge by 1.

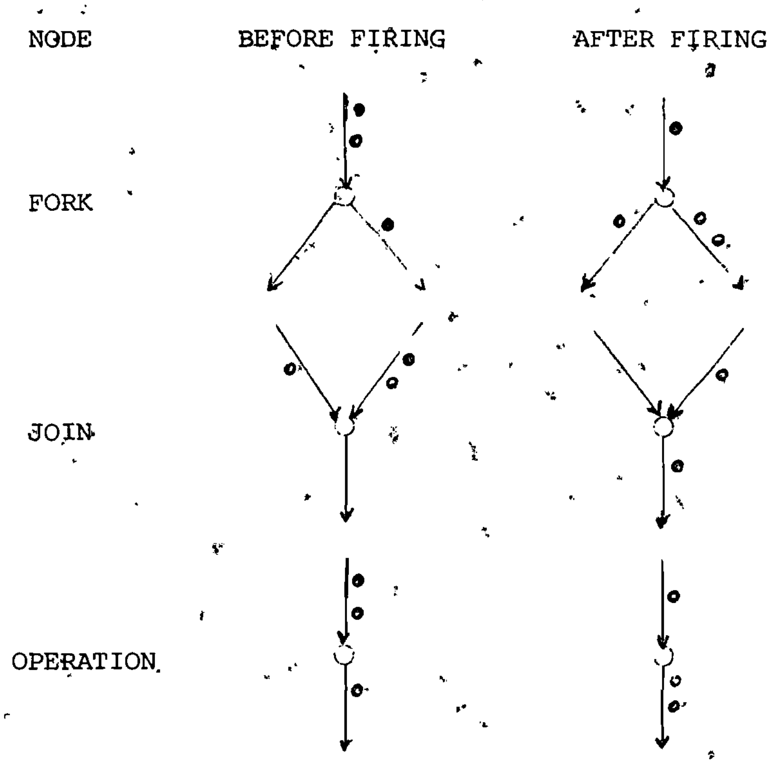


Figure 4.4 - Examples of "firings"

For example, node J in Fig. 4.3 (a) is enabled. The firing of J yields the marked S-graph of Fig. 4.3 (b).

5. SYNCHRONIZATION STRUCTURES

We are interested in synchronization graphs which correspond, in a rather evident way (cf. Section 6), to task flow charts. Such S-graphs form a special class, called synchronization structures. They will be defined in this section (cf. [GI-YO], Section 3).

Let m and m' be markings of the S-graph Γ. We write $m \overset{v}{\rightarrow} m'$ to indicate that the marking m' is obtainable from the marking m by firing node v. We write $m \rightarrow m'$ to state that m' is obtainable from m by the successive firing of one or more nodes of Γ. Furthermore, we set

$$[m] = \{m' \mid m \rightarrow m'\} \cup \{m\}.$$

We shall refer to $[m]$ as the set of all markings reachable from m.

We denote by e_S the outedge of the START node S, and by e_H the inedge of the HALT node H.

Definition The initial marking m_o of an S-graph Γ is defined as follows:

$$m_o(e_S) = 1 \quad \text{and} \quad m_o(e) = 0 \quad \text{for every} \quad e \neq e_S.$$

A marking m of Γ is final iff $m(e_H) > 0$. We denote by M_F the set of all final markings of Γ.

Definition An S-graph is terminating iff

$$(\forall m \in [m_o])([m] \cap M_F \neq \emptyset),$$

i.e. if m is reachable from m_o, then there exists a final marking reachable from m.

Definition Let Γ be an S-graph and E its edge set. Γ is residue-free iff

$$(\forall m \in [m_o])\left[m \in M_F \rightarrow \sum_{e \in E} m(e) = 1\right],$$

i.e. for any final marking m reachable from m_o, the marked S-graph (Γ, m) contains exactly one token (namely on e_H).

Definition. An S-graph Γ is well-formed iff Γ is both terminating and residue-free.

The S-graph Γ_1 of Fig. 4.2 (a) is well-formed, whereas the S-graph Γ_2 of Fig. 4.2 (b) is not terminating. We shall refer to well-formed S-graphs as synchronization structures or S-structures.

Definition. An S-graph Γ with edge set E is safe iff

$$(\forall m \in [m_o])(\forall e \in E)m(e) \leq 1,$$

i.e. the number of tokens on any edge e cannot exceed 1, under any marking m reachable from m_o.

The following result is an immediate consequence of Theorem 3.1 in [YO-GI].

Proposition Every S-structure is safe.

Furthermore, we have

Theorem 5.1 An S-graph is well-formed iff it is cycle-free.

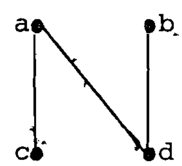
Proof See proof of Theorem 3.1 in [GI-YO].

6. S-STRUCTURES AND POSETS

In this section we establish the relationship between S-structures and task flow charts. Task flow charts describe a partially ordered set (poset) of subtasks. The relationship between S-structures and their corresponding task flow charts is established by means of the following definitions and Theorem 6.1.

<u>Definition</u> A <u>task flow chart</u> is a partially ordered set of tasks.

<u>Definition</u> Let Γ be an S-structure with a nonempty set Σ of OPERATION nodes. With such an S-structure Γ we associate the poset (partially ordered set) $G(\Gamma) = (\Sigma, \sqsubseteq)$, where $\sqsubseteq$ is a partial order relation on Σ: $x \sqsubset y$ holds iff there exists a directed path in Γ from node x to node y, where $x \neq y$; $x \sqsubseteq y$ holds iff $x \sqsubset y$ or $x = y$. For example, the poset $G(\Gamma_1)$, defined by the S-structure Γ_1 of Fig. 4.2(a) is shown (in the usual way of representing posets) in Fig. 6.1.

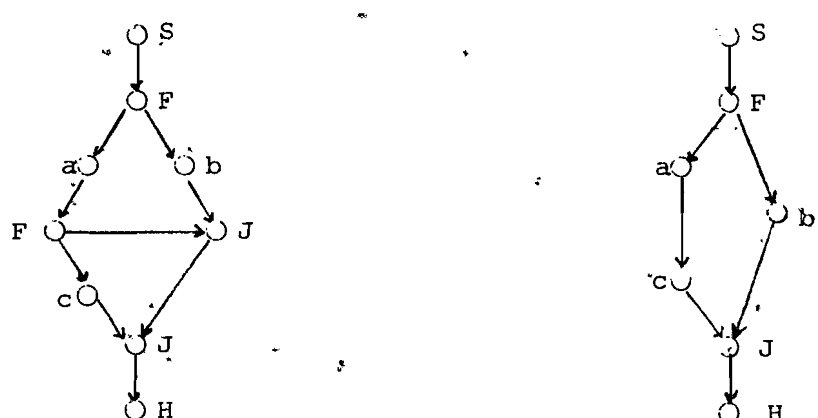


<u>Figure 6.1</u> - Poset $G(\Gamma_1)$ for the S-structure Γ_1 of Fig. 4.2(a).

<u>Theorem 6.1</u> Let $G = (\Sigma, \sqsubseteq)$ be an arbitrary, finite poset. Then there exists an S-structure Γ_G such that $G = G(\Gamma_G)$.

<u>Proof</u>: See proof of Theorem 5.1 in [GI-YO].

It is important to notice that the same poset G may correspond to different S-structures. An example is shown in Fig. 6.2



(a) Γ (b) Γ'

<u>Figure 6.2</u> - Two S-structures Γ and Γ' with equal posets: $G(\Gamma) = G(\Gamma')$.

7. REFINEMENTS AND STRUCTURABILITY

In this section we formalize and extend the concept of "structurable", discussed informally in Section 2.

Structured programming (cf. [DA-DI-HO], [LE-MA]) is based on a suitably selected set Δ of control primitives. A complex program is derived top-down, by "stepwise refinement", involving the set Δ only. The control primitives are "irreducible", i.e. they cannot be obtained from other primitives by refinement. All these concepts, related to S-structures, will now be introduced in a rigorous way.

Definition Let Γ_1 and Γ_2 be S-structures, each containing more than one OPERATION node, and let a be an OPERATION node of Γ_1. Assume $(\Sigma_1 - \{a\}) \cap \Sigma_2 = \emptyset$. Then the refinement $\Gamma = \Gamma_1(a \leftarrow \Gamma_2)$ of Γ_1 is the S-structure Γ defined as indicated in Fig. 7.1.

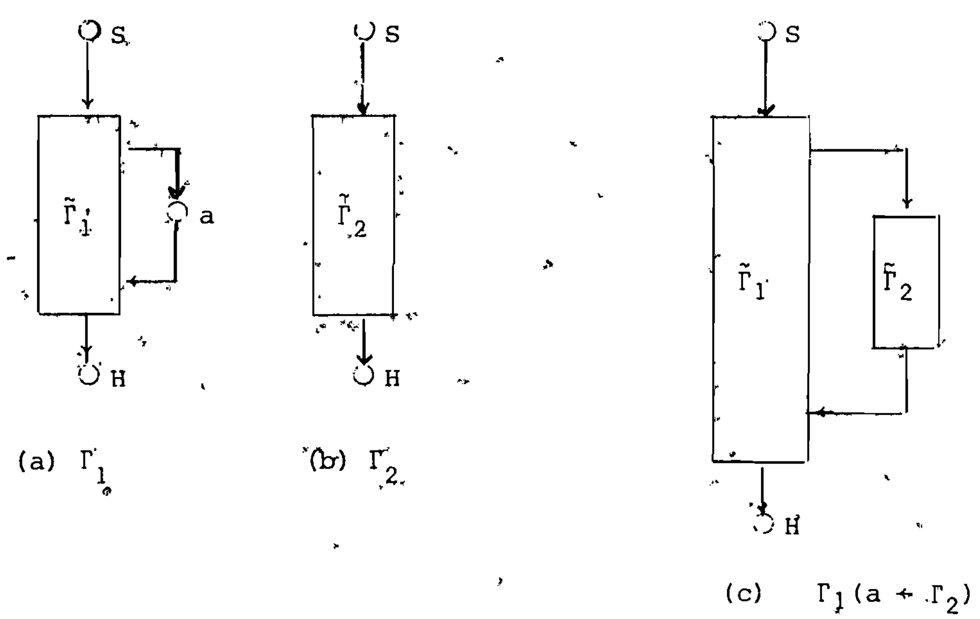


Figure 7.1 - Illustrating the concept of refinement
(a) S-structure Γ_1
(b) S-structure Γ_2
(c) Refinement $\Gamma = \Gamma_1(a \leftarrow \Gamma_2)$

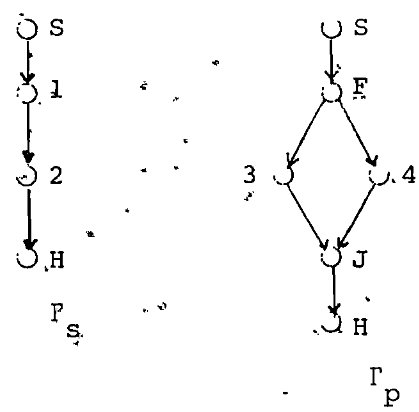
Let Δ be a set of S-structures. A "structuring" with respect to Δ is a finite sequence of S-structures $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$ such that

$\Gamma_1 \in \Delta$ and for every i, $1 \le i < n$, $\Gamma_{i+1} = \Gamma_i(a_i \leftarrow \Gamma^{(i)})$ where a_i is an OPERATION node of Γ_i and $\Gamma^{(i)} \in \Delta$.

<u>Definition</u> The S-structure Γ is Δ-<u>structurable</u> iff there exists a structuring $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$ with respect to Δ, such that $G(\Gamma) \cong G(\Gamma_n)$, i.e. $G(\Gamma)$ can be obtained from $G(\Gamma_n)$ by a relabeling of the nodes.

A set Δ of S-structures is <u>primitive</u>, iff no $\Gamma \in \Delta$ is $(\Delta-\{\Gamma\})$-structurable.

Usually, one considers the primitive set $\Delta_2 = \{\Gamma_s, \Gamma_p\}$ shown in Fig. 7.2.



<u>Figure 7.2</u> - The primitive set $\Delta_2 = \{\Gamma_s, \Gamma_p\}$

It is easily seen that a task flow chart can be composed from its elementary tasks by successively applying the operators '$\|$' and ';' (see Section 2) iff the corresponding S-structures are Δ_2-structurable.

We shall now show that the S-structure Γ_1 of Fig. 4.2(a) is not Δ_2-structurable. Indeed, assume first that such a structuring starts with Γ_s (see Fig. 7.2) and ends with Γ, where $G(\Gamma) \cong G(\Gamma_1)$. Then the operation nodes a,b,c,d of Γ must have a partition into two disjoint subsets Σ_1, Σ_2 such that the nodes of Σ_i $(i = 1,2)$ are the descendants of node i in Γ_s. Furthermore, every node in Σ_1 must precede every node in Σ_2 in $G(\Gamma)$.

But no such partition can yield exactly the above poset $G(\Gamma) = G(\Gamma_1)$, shown in Fig. 6.1. (For example, the partition $\Sigma_1 = \{a,b\}$, $\Sigma_2 = \{c,d\}$ yields $b \sqsubset c$, which is not the case in $G(\Gamma_1)$.) Similarly, one shows that the structuring in question cannot start with Γ_p. This confirms our claim in Section 2, that the task flow chart of Fig. 2.2 is not structurable (with respect to '$\|$' and ';').

It follows that $\Delta_3 = \{\Gamma_s, \Gamma_p, \Gamma_1\}$ is a primitive set. Moreover, we show in the next section that given any finite, primitive set Δ, there exist S-structures which are not Δ-structurable.

8. IRREDUCIBLE S-STRUCTURES

Definition A poset G is reducible iff there exists an S-structure Γ such that $G = G(\Gamma)$ and Γ can be obtained as a refinement. Otherwise, G is irreducible. An S-structure Γ is said to be irreducible iff $G(\Gamma)$ is irreducible.

Thereom 8.1 The posets $G(C_n)$ shown in Fig. 8.2 are irreducible for every $n \geq 1$. Thus the S-structures C_n of Fig. 8.1 are irreducible, for every $n \geq 1$.

Proof See [GI-YO], Section 8.

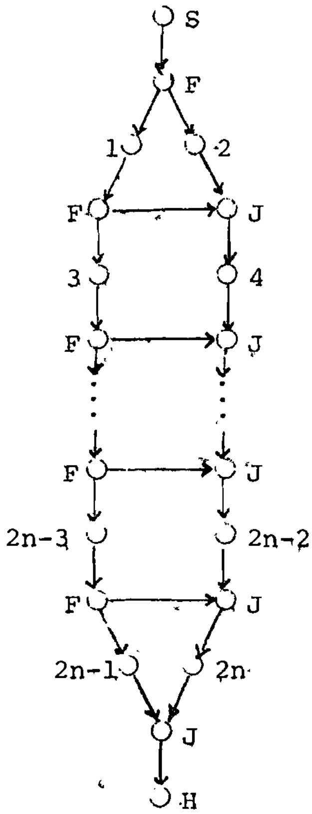


Figure 8.1 - S-structure C_n

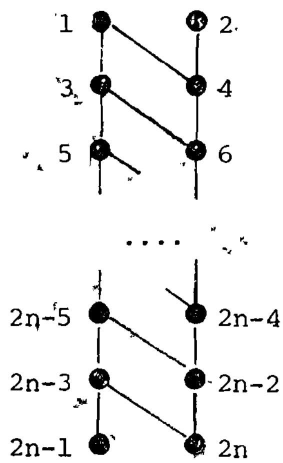


Figure 8.2 - $G(C_n)$

Another infinite family of irreducible posets is shown in Fig. 8.3.

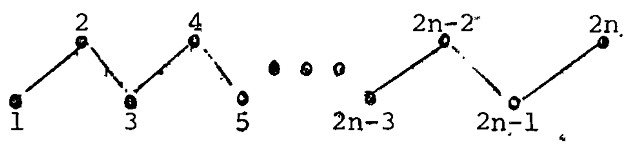


Figure 8.3 - Poset K_n

Indeed, we have

Theorem 8.2 The posets K_n of Fig. 8.3 are irreducible, for every $n \geq 1$.

Thus the S-structures Γ_{K_n}, $n \geq 1$, are all irreducible. Γ_{K_3} is shown in Fig. 8.4.

Proof See proof of Proposition 8.1 in [GI-YO].

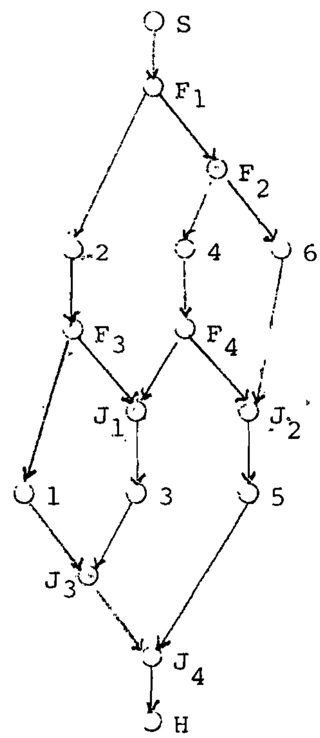


Figure 8.4 - Γ_{K_3}

It follows directly from the above definitions, that an irreducible S-structure Γ is not Δ-structurable, for any Δ which does not contain an S-structure Γ' such that $G(\Gamma) \cong G(\Gamma')$.

Hence the following result holds.

Corollary: Given any finite set Δ of S-structures, there exists an S-structure Γ which is not Δ-structurable.

REFERENCES

[BR-YO] J.A. Brzozowski, and M. Yoeli, Digital Networks, Prentice-Hall, 1976.

[BRU-ALT] J. Bruno, and S.M. Altman, "A Theory of Asynchronous Control Networks", IEEE Trans. Comp., Vol. C-20, June 1971, pp. 629-638.

[CO-MA] S.E. Conry and R.M. Mattheyses, "On Construction of Control Networks for Parallel Systems", Proc. 1st Europ. Conference on Parallel and Distributed Processing, Toulouse, Feb. 14-16, 1979, pp. 170-179.

[DA-DI-HO] W.J. Dahl, E.W. Dijkstra and C.A.H. Hoare, Structured Programming, Academic Press, New York, 1972.

[GI-YO] A. Ginzburg, and M. Yoeli, "Reducibility of Synchronization Structures", TR #350, Dept. of Computer Science, Technion, Haifa, June 1983.

[HE-YO] O. Herzog, and M. Yoeli, "Control Nets for Asynchronous Systems", Part I, TR #74, Dept. of Computer Science, Technion, Haifa, May 1976.

[KEL] R.M. Keller, "Towards a Theory of Universal Speed-Independent Modules", IEEE Trans. Comp., Vol. C-23, January 1974, pp. 21-33.

[LE-MA] H.F. Ledgard, and M. Marcotty, "A Genealogy of Control Structures", Comm. ACM, 18, (1975), pp. 629-639.

[VAL] R. Valette, "Sur la Déscription, l'Analyse, et la Validation des Systèmes de Commande Parallèles", D.Sc. Thesis, University Paul Sabatier, Toulouse, 1976.

[WE-SM] P. Wegner, and S.A. Smolka, "Processes, Tasks, and Monitors: A Comparative Study of Concurrent Programming Primitives", IEEE Trans. Softw. Eng., Vol. SE-9, July 1983, pp. 446-462.

[WEI] C. Weitzman, Distributed Micro/Minicomputer Systems, Prentice-Hall, 1980.

[YO-GI] M. Yoeli and A. Ginzburg, "Control Nets for Parallel Processing", Inf. Processing 80, Proc. 8th World IFIP Congress 80, (North-Holland, Amsterdam, 1980), pp. 71-76.

[YOE] M. Yoeli, "A Structured Approach to Parallel Programming and Control," Proc. 1st Europ. Conference on Parallel and Distributed Processing (1979), pp. 163-169.