A STRUCTURED APPROACH TO PARALLEL
PROGRAMMING AND CONTROL

by
Michael Yoeli

Technical Report #126
April 1973
1. INTRODUCTION

Structured top-down programming has become an important methodology for the design of correct computer programs. In this paper we set up a mathematical framework by means of which this structured approach may be extended to the systematic design of parallel computation structures. The paper may also be viewed as an attempt to develop an algebraic theory of structured parallel programming based on presently available algebraic theories of sequential flowcharts ([KN-FL],[KOS],[MIL],[LE-MA],[ELG]) as well as proposed models of parallel computation structures ([HE-YO], [YO-BR],[VAL]).

2. PARALLEL CONTROL SCHEMATA

Many digital systems may be considered as consisting of two parts: a device structure and a control structure [BR-YO],[YO-BR]. The device consists of specific devices, such as registers, adders, counters, etc. The control structure supervises the activities and sequencing of these devices.

We assume all devices to operate asynchronously. Such a device is given a GO command by the control structure to start its operation. Upon completion of its task the device returns a DONE signal. In this paper we are not concerned with the various methods by which control commands and signals may be realized (see [BR-YO],[YO-BR]).

We assume that each device or operational unit performs some specific task (e.g. addition). In order to specify the sequence in which the various operational units are to perform their task, we introduce a suitable
extension of the concept of flowchart, namely parallel control schema (PCS).

A parallel control schema (PCS) $S$ consists of the following:

(1) A finite, directed graph $G(S)$, the nodes of which are partitioned into seven types, as indicated in Figure 1.

(2) A finite alphabet $\Sigma$ of operation letters. Every OPERATION node of $G(S)$ is labeled by a letter of $\Sigma$.

(3) A finite alphabet $\Pi$ of predicate letters. Every DECIDER node $D$ of $G(S)$ is labeled by a letter of $\Pi$. Furthermore, one outgoing edge of $D$ is labeled $T$ (true), and the other edge $F$ (false).

<table>
<thead>
<tr>
<th>NODE TYPE</th>
<th>INDEGREE</th>
<th>OUTDEGREE</th>
<th>GRAPHICAL REPRESENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>HALT</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DECIDER</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>UNION</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>FORK</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JOIN</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>OPERATION</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Node types of parallel control schemata
Every PCS $S$ satisfies the following conditions:

(a) $S$ has exactly one START node and exactly one HALT node.
(b) if $v$ is a node of $S$, and $v \neq \text{START}$, then there exists a directed path from START to $v$.
(c) If $v$ is a node of $S$, and $v \neq \text{HALT}$ then there exists a directed path from $v$ to HALT.
(d) $S$ has no multiple edges.

One easily verifies that $S$ contains no self-loops (i.e. cycles of length 1). The concept of PCS introduced here is a slight modification of the concept of control net defined in [HE-YO]. An example of a PCS is shown in Figure 2.

A formal definition of the control dynamics of a PCS by means of Petri nets will be given in Section 6. Informally, DECIDER and UNION nodes correspond to predicate and collecting nodes of conventional flowcharts [MIL] respectively. Thus, a DECIDER directs control from its in-edge to one of its two out-edges. A UNION passes control from one of its in-edges to its out-edge. On the other hand, a FORK passes control from its in-edge to both of its out-edges. A JOIN passes control to its out-edge, provided control signals appear on both its in-edges. To start the activity of a PCS, the START node issues a control signal. The activity terminates, whenever a control signal arrives at the HALT node.

For any operation letter $\sigma_k \in \Sigma$ let us denote by $k$ the initiation of the corresponding operation, and by $k$ its termination. Then a possible activity of the PCS of Figure 2 is represented by the activity sequence $3 1 2 6 3 1 2 4 5 6 3$. 

In Section 6 we associate with any given PCS $S$ its **extended language** $L(S)$, consisting of all possible finite and infinite activity sequences of $S$. Two PCSs $S_1$ and $S_2$ are defined to be **L-equivalent** iff $L(S_1) = L(S_2)$. The PCS of Figure 2 is L-equivalent to the PCS of Figure 3. This latter PCS is **structured** in a sense to be explained in the next section.
Figure 3: A structured PCS L-equivalent to the PCS of Figure 2.
3. STRUCTURED PARALLEL CONTROL SCHEMATA

We call the graph $G(S)$ of any PCS $S$ a Parallel Control Graph (PCG). Let $G_1$ and $G_2$ be PCGs and $v$ an OPERATION node of $G_1$. We define $G_1(v + G_2)$ to be the PCG $G$ obtained by substituting $G_2$ for $v$ in $G_1$, as indicated in Figure 4.

![Diagram of Parallel Control Graphs](image)

Figure 4: Substitution of PCGs
If \( S \) denotes the PCS of Figure 3, then \( G(S) \) can be obtained from the tree primitive PCGs of Figure 5 by suitable successive substitution, as one easily verifies. Consequently, there exists a structured representation (namely \( S \)) of the PCS of Figure 2 (which is L-equivalent to \( S \)) with respect to the primitives of Figure 5.

An essential problem of structured parallel programming is the selection of suitable sets of "primitive" PCGs. We shall return to this problem in Section 8, after introducing a suitable mathematical formulation of the various concepts discussed so far (see also [VAL]).

In Section 7 we introduce the concept of well-formed PCG (following [HE-YO]) and provide a precise formulation of the statement that "structured parallel programming preserves the property of being well-formed".
4. MIXED-TYPE PETRI NETS

An extensive literature is presently available demonstrating the suitability of Petri nets or related concepts to the modelling of discrete-event systems involving parallel processing [YOE, PET, VAL, AS-VA-OI].

We introduce in this section a modified version of Petri nets which will be applied in Section 6 to a precise formulation of the concept of PCS.

In [YOE] a distinction was made between Boolean-type Petri Nets (BPN) and Integer-type Petri Nets (IPN). Although such a distinction is important for the proper modelling of discrete systems, the literature on Petri nets deals almost exclusively with IPNs. Following [WEN] and [YO-BA] we now introduce the concept of Mixed-type Petri Net (MPN), of which BPNs and IPNs are special cases.

A Mixed-Type Petri Graph (MPG) is a system

\[ G = \langle P, T, R, B \rangle \]

where

- \( P \) is a finite set of places
- \( B \) is a subset of \( P \) (elements in \( B \) are Boolean-type places, elements in \( P-B \) are Integer-type places)
- \( T \) is a finite set of transitions, disjoint from \( P \).
- \( R \) is a binary relation on \( P \cup T \) satisfying the condition
  \[ R \subseteq (P \times T) \cup (T \times P). \]

A Mixed-Type Petri Net (MPN) is an ordered pair \( \langle G, m \rangle \) where \( m \), the marking, is a function \( m: P \rightarrow \mathbb{N} \) (\( \mathbb{N} \) denotes the set of non-negative integers) satisfying the condition \( \{ m(p) | p \in B \} \subseteq \{0,1\} \).
MPNs are conveniently represented by diagrams where a place \( p \in B \) is shown as a circle \( \bigcirc \), a place \( p \in P-B \) as a double circle \( \bigcirc \), and a transition as a bar \( \mid \) (see Fig. 6).

A directed edge from a place \( p \) to a transition \( t \) indicates that \( p R t \), i.e. \( (p,t) \in R \), and a directed edge from \( t \in T \) to \( p \in P \) indicates that \( t R p \). The marking \( m \) is shown by placing \( m(p) \) dots (tokens) into the circle (or double circle) representing \( p \). Alternatively, the integer \( m(p) \) may be written into the double circle representing the integer-type place \( p \).

The place \( p \) is an input place of the transition \( t \) iff \( p R t \) and an output place of \( t \), iff \( t R p \).

Let \( \langle G,m \rangle \) be a MPN. The transition \( t \) is firable iff \( m(p) > 0 \) for every input place \( p \) of \( t \).

If \( t \) is firable, the firing of \( t \) consists in decreasing the marking \( m(p) \) of its input places \( p \) by 1, and replacing the marking \( m(q) \) of its output places \( q \) by \( m(q)+1 \), where + denotes Boolean addition, if \( q \in B \), and integer-addition, if \( q \in P-B \).

In the diagram of Figure 6(a), both transitions \( t_1 \) and \( t_2 \) are firable. If \( t_1 \) fires, the MPN of Figure 6(b) is obtained, whereas the firing of \( t_2 \) yields the MPN of Figure 6(c).

Note that the transition \( t_2 \) is no longer firable in Figure 6(b). Similarly, the firing of \( t_2 \) in the MPN of Figure 6(a) destroys the firable of \( t_1 \). Such a situation is referred to as conflict.

Petri nets are intended to model discrete-event systems. A Boolean-type place \( p \) represents a condition, which holds iff \( m(p) = 1 \). An
Figure 6: (a) Diagram of a Mixed-type Petri Net (MPN).
(b) The MPN obtained after $t_1$ fires.
(c) The MPN obtained after $t_2$ fires.
Integer-type place may be interpreted as representing a counter. Firable transitions correspond to events which may occur under the prevailing conditions. We shall assume that events occur instantaneously, and that only one event may occur at a time.

5. LABELED MIXED-TYPE PETRI NETS

A Labeled Mixed-Type Petri Net (LMPN) \( \Gamma \) consists of the following:

1. A Mixed Petri Graph (MPG) \( G(\Gamma) = \langle P, T, R, B \rangle \).
2. An initial marking \( m_0 \) and a final marking \( m_F \) of \( G \).
3. A finite alphabet \( \Sigma(\Gamma) \) together with a mapping \( w: T \to \Sigma(\Gamma) \cup \{\lambda\} \), where \( \lambda \) denotes the empty string.

Let now \( \Gamma \) be an LMPN as defined above. We write \( m_1[t] > m_2 \) to state that the transition \( t \in T \) is firable in \( <G(\Gamma), m_1> \) and that \( m_2 \) is the marking obtained by firing \( t \).

If there exists a finite firing sequence

\[
m_1[t_1] > m_2, m_2[t_2] > m_3, \ldots, m_{k-1}[t_{k-1}] > m_k
\]

we write \( m_1[s] > m_k \), where

\[
s = (t_1, \ldots, t_{k-1}).
\]

Also, we denote by \( w(s) \) the string in \( \Sigma^* \) defined by

\[
w(s) = \prod_{i=1}^{k-1} w(t_i)
\]

where \( \prod \) denotes concatenation.
With any given LMPN $\Gamma$ we associate its finite-string language $L_f(\Gamma)$ defined by

$$L_f(\Gamma) \triangleq \{w(s) \mid m_0[s] \# m_f\}.$$

As to available studies of Petri net languages, defined similarly, we refer the reader to [HACK], [PET].

Consider now an infinite firing sequence of $\Gamma$,

$$m_1[t_1> m_2, \ldots, m_i[t_i> m_{i+1}, \ldots.$$  

We write $m_1[\alpha>$, where $\alpha$ denotes the infinite sequence

$$\alpha = t_1, t_2, \ldots, t_i, \ldots.$$  

With $\alpha$ we associate the $\Sigma$-sequence

$$w(\alpha) = \prod_{i=1}^{\infty} w(t_i) \in \Sigma^* \cup \Sigma^\omega$$

where $\Sigma^\omega$ denotes the set of all infinite sequences ($\omega$-length strings) over $\Sigma$.

Furthermore, we associate with $\Gamma$ its extended language $L(\Gamma)$ defined by

$$L(\Gamma) = L_f(\Gamma) \cup \{w(\alpha) \mid m_0[\alpha>\}.$$  

We refer the reader to [EIL] for a discussion of $\omega$-length strings.
6. PARALLEL CONTROL SCHEMATA VIEWED AS LABELED PETRI NETS

Let $S$ be a PCS, as defined in Section 2. By applying the transformation rules summarized in Figure 7, we obtain the LMPN $\Gamma(S)$ associated with $S$. In this LMPN all places are Boolean, and $\Sigma(\Gamma(S)) = \overline{\Sigma} \cup \Sigma \Pi \cup \overline{\Pi}$ where $\overline{\Sigma} = \{ \overline{\sigma} | \sigma \in \Sigma \}$, $\Sigma = \{ \sigma | \sigma \in \Sigma \}$, and $\overline{\Pi} = \{ \overline{p} | p \in \Pi \}$.

We define the extended language $L(S)$ of $S$ by $L(S) \triangleq L(\Gamma(S))$. As mentioned in Section 2, the PCSs $S_1$ and $S_2$ are $L$-equivalent iff $L(S_1) = L(S_2)$. The reader will easily verify that $L(S)$ indeed consists of all possible activity sequences, as discussed informally in Section 2.

To simplify the representation of $L(S)$ for a given PCS $S$, we introduce the following notations.

Let $\Sigma$ be a finite alphabet. For $x,y \in \Sigma^*$ we define (see [HACK]) $(x \parallel y) \subseteq \Sigma^*$ recursively as follows:

(a) $\emptyset \parallel \lambda = \lambda \parallel \emptyset = \{ \emptyset \}$ for every $\sigma \in \Sigma$.

(b) For $\sigma x, \sigma \in \Sigma, x \in \Sigma^*$, and $\tau y, \tau \in \Sigma, y \in \Sigma^*$ we set

$$\sigma x \parallel \tau y = (\{\sigma\},(x \parallel \tau y)) \cup (\{\tau\},\sigma x \parallel y)$$

where

$$(X,Y) \triangleq \{xy \mid x \in X \land y \in Y\}.$$ 

For $X \subseteq \Sigma^*$ we set

$$X^\omega \triangleq \{ \prod_{i=0}^{\infty} x_i \mid \forall i \in \mathbb{N} : x_i \in X \}.$$ 

For example, for the PCS $S$ of Figure 3 we have

$$L(S) = A(B^* + B^{\omega}).$$
where  

\[ A = (\hat{\sigma}_1 \hat{\sigma}_3) \parallel (\hat{\sigma}_2 \hat{\sigma}_4) \]

\[ B = \hat{\sigma}_5 (\hat{\sigma}_6 \parallel \hat{\sigma}_4) \]

\[ \hat{\sigma}_k = \sigma_k \]

One easily verifies that the PCSs of Figure 2 and Figure 3 are L-equivalent, as mentioned in Section 2. Other examples of L-equivalent PCSs are shown in Figure 8 and Figure 9. Figure 8(a) illustrates the well-known structured representation of the "DOUNTL" schema of Figure 8(b), using the primitive PCGs of COMPOSITION (Figure 5(a)) and DOWHILE (Figure 5(b)).

Let \( S_a \) and \( S_b \) be the PCS of Figure 5(a) and (b), respectively. Then

\[ L(S_a) = \hat{\sigma} [(p\,\hat{\sigma})^\times \hat{\sigma} + (p\,\hat{\sigma})^\omega] \]

\[ L(S_b) = (\hat{\sigma} p)^\times \hat{\sigma} \hat{\sigma} + (\hat{\sigma} p)^\omega \]

where \( \hat{\sigma} = \sigma \).

Evidently, \( L(S_a) = L(S_b) \).

Similarly, the PCSs of Figure 9, illustrating "node-splitting", are L-equivalent. For other examples of node-splitting, see [KOS].
Figure 8: Example of L-equivalent PCSs.

Figure 9: Example of node splitting.
7. WELL-FORMED CONTROL SCHEMATA

The concept of well-formed control net introduced in [HE-YO] is also applicable to PCSs.

Let \( \Gamma \) be an LMPN, as defined in Section 5; For any marking \( m \) of \( \Gamma \) we let \([m]\) denote the set of all markings reachable from \( m \), including \( m \). Thus,

\[ [m] \triangleq \{ m' \mid m >^s m' \text{ for some } s \in T^+ \} \cup \{ m \}. \]

The LMPN \( \Gamma \) is live iff

\[ (\forall m \in [m_o]): m_F \in [m]. \]

The LMPN \( \Gamma \) is residue-free iff

\[ (\forall m \in [m_o]): m \geq^s m_F \rightarrow m = m_F, \]

where \( m \geq^s m' \equiv (\forall p \in P): m(p) \geq^s m'(p). \)

The LMPN \( \Gamma \) is \( k \)-safe iff

\[ (\forall m \in [m_o]) (\forall p \in P): m(p) \leq^s k. \]

A PCS \( S \) is live (residue-free), iff \( \Gamma(S) \) is live (residue-free).

A PCS \( S \) is safe, iff the LMPN obtained from \( \Gamma(S) \) by replacing all its places by integer-type places is 1-safe. A PCS \( S \) is well-formed, iff it is live, safe, and residue-free. In [HE-YO] necessary and sufficient conditions are given for a control net to be well-formed. These conditions also apply to PCSs.

Clearly, the property of a PCS \( S \) to be well-formed, depends only on \( G(S) \). Hence, if the PCS \( S \) is well-formed, we also say that \( G(S) \) is well-formed.
Let now $\Delta$ be a set of PCGs. We say that the PCS $S$ is covered by $\Delta$, iff there exists a PCS $S'$, L-equivalent to $S$, such that $G(S')$ can be obtained from $\Delta$ by successive substitutions; i.e. there exists a finite sequence

$$G_0, G_1, \ldots, G_k$$

such that $G_0 \in \Delta$, $G_k = G(S')$, and

$$G_{i+1} = G_i(v_i + G_\Delta) \quad \text{for} \quad 0 \leq i < k$$

where $G_\Delta \in \Delta$.

The following results are easily verified:

(A) Let $G_1$ and $G_2$ be well-formed PCGs. Then $G_1(v + G_2)$ is well-formed.

(B) Let $\Delta$ be a set of well-formed PCGs. Let $G_k$ be obtainable from $\Delta$ by successive substitutions, as specified above. Then $G_k$ is well-formed.

8. PCS - PRIMITIVES

A variety of flowchart primitives have been proposed in connection with structured (sequential) programming and their hierarchies have been established, based on various definitions of equivalence ([KOS], [LE-MA], [ELG]). Very little is known as to the analogous problems related to structured parallel programming, and much further research is required in this area. We formulate in this section a conjecture, which, if proved correct, would be very useful in establishing suitable PCS-primitives.
We call DU-schema any PCS containing only nodes of the following types: START, STOP, DECIDER, UNION, OPERATION. Similarly we call FJ-schema any PCS containing only nodes of the following types: START, STOP, FORK, JOIN, OPERATION.

Conjecture. Let $\Delta_{DU}$ and $\Delta_{FJ}$ be sets of well-formed PCGs, such that any well-formed DU-schema and any well-formed FJ-schema are covered by $\Delta_{DU}$ and $\Delta_{FJ}$, respectively. Then any well-formed PCS is covered by $\Delta_{DU} \cup \Delta_{FJ}$. 
REFERENCES


REFERENCES (cont'd)

