TOWARDS A CHARACTERIZATION OF PROGRAMS
FOR A MODEL OF VLSI ARRAY-PROCESSORS

I.V. Ramakrishnan, D.S. Fussell, A. Silberschatz

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

TR-202 July 1982

---

1This research was supported in part by the National Science Foundation under Grant MCS-8104017.
ABSTRACT

This paper is concerned with understanding the properties of programs correctly executable on a model of linear array processors suitable for VLSI. Such a model has been proposed as an attractive architecture to handle compute-bound problems in an efficient and cost-effective manner. We provide a complete syntactic characterization for a class of programs that are correctly executable on this model and show that among the class of syntactically characterized programs the sub-class of programs correctly executable without semantics is very limited. We then identify the semantic properties required of programs in this entire syntactic class inorder for them to be correctly executable.
1. Introduction

Interests in parallel processing were created by the emergence of parallel computers namely ILLIAC. The recent advent of large-scale integration technology has further stimulated interests in parallel processing [2, 3, 4, 6, 8]. VLSI offers the potential of realizing parallel computations in silicon. Such a realization can be made cost-effective and modular provided:

1. The processors used are simple and uniform.

2. The processors are connected by a modular, simple and regular interconnection network and this implies that programs executed on such a network exhibit simple and regular data and control flow.

The realization is rendered efficient by extensive use of pipelining and multiprocessing. These notions of simplicity and regularity will be made more precise in the subsequent sections.

In this paper we have characterized the properties of programs correctly executable on a model of linear array processors for VLSI. We have distinguished the roles of two types of properties in a correct mapping (execution) of programs onto such processor-arrays, namely-

1. Syntactic - i.e., the structure of programs

2. Semantic - i.e., some knowledge of what the programs do

The main results in this paper are lemma 4.0-5, theorem 5.1-1, theorem 5.2-1 and theorem 6.1-1. Our paper is organized as follows:
In section-2 we introduce the program and linear array models. In section-3 we define the problem of correct execution of programs on a linear array. In section-4 and section-5 we provide a syntactic characterization of correctly executable programs and in section-6 we demonstrate the importance of knowing the semantics of the syntactically characterized programs inorder for them to be correctly executable. In section-7 we illustrate the characterization by synthesizing algorithms for two important computational problems.

2. Program and Linear-Array Models

In this section we will describe our program and linear-array models. We will also make the notions of simplicity and uniformity of processors more precise.

2.1. Graph and Set-Theoretic Preliminaries

Let \( A = \{ a_1, a_2, \ldots, a_n \} \) be a set of elements. Let \( R \) be a binary relation on \( A \). Let \( R^+ \) denote the transitive closure of \( R \).

Definition 2.1.1: \( R \) is total on \( A \) iff for any \( a_i \) and \( a_j \) in \( A \), either \( a_i R^+ a_j \) or \( a_j R^+ a_i \).

Let \( S = \{ s_1, s_2, \ldots, s_m \} \) be a set of binary relations on \( A \).

Let \( G_S = (V, E) \) be the directed graph induced by all the relations in \( S \) on \( A \) with \( V = A \) and \( E = \{ \langle a_i, a_j \rangle \mid a_i s_x a_j \text{ for some } s_x \text{ in } S \} \).

Let \( G_x = (V_x, E_x) \) be the subgraph of \( G_S \) induced by the relation \( s_x \) in \( S \) on \( A \) with \( V_x = V \) and \( E_x = \{ \langle a_i, a_j \rangle \mid a_i s_x a_j \} \).
Definition 2.1-2: \( S \) imposes a consistent order on \( A \) iff

1. there exists at least one relation \( s_x \) in \( S \) such that \( s_x \) is total and \( G_x \) is acyclic.

2. for every \( s_y \) in \( S \) there is a constant \( c_y \) associated with \( s_y \) such that for any \( a_i \) and \( a_j \) in \( A \), if \( a_i \ s_y \ a_j \) then \( \pi_A(a_j) = \pi_A(a_i) + c_j \) where \( \pi_A \) is the indexing function that maps every element in \( A \) to its position in the total order imposed by a topological sort\(^1\) on the vertices in \( C_x \).

We will call \( c_y \) the consistency constant of \( s_y \).

Definition 2.1-3: A relation \( s_x \) in \( S \) imposes a linear chain on \( A \) iff \( S \) imposes a consistent order on \( A \) and \( c_x=1 \).

Example 2.1-1: Let \( A=\{a_1, a_2, a_3, a_4\} \) and \( S=\{s_1, s_2\} \) such that \( a_2 \ s_1 \ a_1, a_1 \ s_1 \ a_4, a_4 \ s_1 \ a_3, a_2 \ s_2 \ a_4 \) and \( a_1 \ s_2 \ a_3 \). Set \( s_x=s_1 \). \( G_S \) and \( G_x \) are

![Diagram of \( G_S \) and \( G_x \)]

Figure 2.1-1: \( G_s \) Figure 2.1-2: \( G_x \)

The indexing function \( \pi_A \) is \( \pi_A(a_2)=1, \pi_A(a_1)=2, \pi_A(a_4)=3 \) and \( \pi_A(a_3)=4 \). The consistency constants for \( s_1 \) and \( s_2 \) are 1 and 2 respectively. \( S \)

\(^1\)In this paper we will be assuming that if \( V \) is a set of vertices then the total order imposed by a topological sort on \( V \) is a set of consecutive integers ranging from 0 to \(|V|-1\).
imposes a consistent order on \( A \) and \( s_1 \) imposes a linear chain on \( A \).

**Example 2.1-2:** Let \( A, s_1, \) and \( s_2 \) be defined as in example 2.1-1. Let \( S = \{ s_1, s_2, s_3 \} \) where \( a_2, s_3, a_3 \) and \( a_1, s_3, a_4 \). \( G_S \) is shown in figure 2.1-3.

![Figure 2.1-3](image)

\( S \) does not impose a consistent order on \( A \) as \( s_3 \) does not have a consistency constant.

**Definition 2.1-4:** Consider a labelled directed graph \( G = \langle V_G, E_G, S_{O_G}, S_{I_G}, L_G, G_E, G_{I_G} \rangle \) where:

1. \( V_G, S_{O_G} \) and \( S_{I_G} \) are three distinct sets of vertices with \( S_{O_G} \) as the set of source vertices, \( S_{I_G} \) as the set of sink vertices and \( V_G \) as the set of remaining vertices called computation vertices.

2. \( E_G \) is a set of edges

3. \( L_G \) is a set of labels

4. \( G_E \) and \( G_{I_G} \) are two many-one functions such that:

   a. \( G_E : E_G \longrightarrow L_G \)
b. \( G_{10} : \text{SI}_G \cup \text{SO}_G \rightarrow L_G \)

\( G \) is a uniform graph iff it satisfies the following properties:

1. Every vertex in \( \text{SO}_G \) (\( \text{SI}_G \)) has exactly one edge directed from (to) it to (from) some vertex in \( V_G \). The labels of any vertex in \( \text{SO}_G \) (\( \text{SI}_G \)) and the edge directed from (to) it are the same.

2. For any vertex in \( V_G \) all the edges directed to (from) the vertex have distinct labels and the number of edges directed to (from) the vertex is equal to \( |L_G| \).

Example 2.1-3:

![Diagram of a uniform graph]

**Figure 2.1-4: Uniform Graph**

\( L_G = \{11, 12\} \) and \( V_G = \{v_j \mid 1 < j < 3\} \).

\( E_{11} = \{e_{j1} \mid 1 < j < 5\} \), \( E_{12} = \{e_{j12} \mid 1 < j < 4\} \) and \( E_G = E_{11} \cup E_{12} \).

\( \text{SI}_{11} = \{ij_{11} \mid 1 < j < 2\} \), \( \text{SI}_{12} = \{i_{12}\} \) and \( \text{SI}_G = \text{SI}_{11} \cup \text{SI}_{12} \)

\( \text{SO}_{11} = \{oj_{11} \mid 1 < j < 2\} \), \( \text{SO}_{12} = \{o_{12}\} \) and \( \text{SO}_G = \text{SO}_{11} \cup \text{SO}_{12} \).

The label of edges in \( E_{11} \), the vertices in \( \text{SO}_{11} \) and \( \text{SI}_{11} \) is 11. The
label of edges in \( E_{12} \), the vertices in \( S_{012} \) and \( S_{112} \) is 12.

Henceforth G will always denote a uniform graph.

Definition 2.1-5: For any label 1 in G, a major path labelled 1 is a directed path from a source vertex \( v_x \) to a sink vertex \( v_y \) such that the label of \( v_x \), \( v_y \) and all the edges in the path is 1.

Let \( r_1 \) be a binary relation on major paths where 1 is some label in G.

Definition 2.1-6: For any pair of major paths \( q_p \) and \( q_r \) in G, \( q_p \ r_1 q_r \) iff there exist computation vertices \( v_x \) and \( v_y \) in \( q_p \) and \( q_r \) respectively such that there is an edge labelled 1 directed from \( v_x \) to \( v_y \).

We will illustrate all the above graph-theoretic definitions by two examples.

Example 2.1-4: In example 2.1-3 the major path labelled 11 is the path directed from \( i_{11} \) to \( o_{11} \) through vertices \( v_1 \), \( v_3 \) and \( v_2 \) in that order. The edges in this path are \( e_{11} \), \( e_{21} \), \( e_{31} \) and \( e_{41} \). There are two major paths labelled 12. One path is directed from \( i_{12} \) to \( o_{12} \) through vertices \( v_1 \) and \( v_2 \) in that order. The edges in this path are \( e_{12} \), \( e_{22} \) and \( e_{32} \). The other major path is the path directed from \( i_{212} \) to \( o_{212} \) through \( v_3 \). The edges are \( e_{42} \) and \( e_{52} \) in this path.

Example 2.1-5:
In figure 2.1-5, \( L_G = \{11, 12\} \), the set of computation vertices \( V_G = \{v_1, v_2, v_3\} \), the set of source vertices \( S0_G = \{i_{11}, i_{12}, i_{212}, o_{312}\} \) and the set of sink vertices \( SL_G = \{o_{11}, o_{12}, o_{212}, o_{312}\} \). The vertices \( i_{11} \) and \( o_{11} \) are labelled 11 and the vertices \( i_{12}, i_{212}, i_{312}, o_{12}, o_{212} \) and \( o_{312} \) are labelled 12. The horizontal edges are labelled 11 and the vertical edges are labelled 12. The major path labelled 11 is the horizontal path and the major paths labelled 12 are the three vertical paths. The relation \( r_{11} \) is non-empty while \( r_{12} \) is empty.

Definition 2.1-7: Two computation vertices \( v_x \) and \( v_y \) are transitively related by an edge \( e_a \) directed from \( v_x \) to \( v_y \) iff there exists a computation vertex \( v_z \) and two edges such that one edge is directed from \( v_x \) to \( v_z \) and the other edge is directed from \( v_z \) to \( v_y \).

Example 2.1-6:
In figure 2.1-6 \( e_a \) relates \( v_x \) and \( v_y \) transitively.

**Definition 2.1-8:** Two major paths \( q_p \) and \( q_r \) are identical iff:

1. the computation vertices in \( q_p \) and \( q_r \) are identical.

2. for any computation vertex \( v_x \) in \( q_p \) and \( q_r \), its index in any topological sort of the vertices in \( q_p \) is the same as its order in any topological sort of the vertices in \( q_r \).

**Example 2.1-7:**

In figure 2.1-7 \( v_1 \), \( v_2 \) and \( v_3 \) are computation vertices, \( i_{11} \) and \( i_{12} \) are source vertices labelled 11 and 12 respectively. \( o_{11} \) and \( o_{12} \) are two sink vertices labelled 11 and 12 respectively. The edges above the
dotted line are labelled 11 and the edges below the dotted line are labelled 12. \( q_p \) is the major path labelled 11 and is directed from 11 to 011 through edges shown above the dotted line. \( q_r \) is the major path labelled 12 and is directed from 112 to 012 through edges shown below the dotted line. \( q_p \) and \( q_r \) are identical.

2.2. Program Model

Let \( Y_1, Y_2, \ldots, Y_k \) be \( n \) sets of scalar elements. We do not wish to provide a formal definition of scalar elements. Suffice it is to say that a scalar element is a typical value held in a memory location of a machine like IBM/360. Let \( Y=Y_1 \times Y_2 \times \cdots \times Y_k \) be the cartesian product of \( k \) sets. Let \( x_i \) denote the \( i \)th component of a tuple \( x \) in \( Y \) and so \( x_i \) is in \( Y_i \).

**Definition 2.2-1:** A program is a one-one function \( \Psi: D \rightarrow R \) where \( D \subseteq Y, R \subseteq Y \) and

1. for any pair of tuples \( x \) and \( z \) in \( D \), \( x_i \neq z_i \)

2. for any pair of tuples \( x \) and \( z \) in \( R \), \( x_i \neq z_i \)

3. for any pair of tuples \( x \) and \( z \) where \( x \) is in \( D \) and \( z \) is in \( R \) and if \( \Psi(x)=z \) then \( x_i \neq z_i \)

[Note: (1), (2) and (3) are not restrictive as multiple occurrences can be replaced by distinct elements.]

Let \( D_i = \{ x_i \mid x_i \text{ is the } i \text{th component of tuple } x \text{ in } D \} \) and \( R_i = \{ x_i \mid x_i \text{ is the } i \text{th component of tuple } x \text{ in } R \} \). Let \( D_i' = \{ x_i \mid x_i \in D_i \text{ and } \} \)
Let \( x_i \in R_1 \) and \( R_1' = \{ x_i \mid x_i \in R_1 \text{ and } x_i \notin D_1 \} \). Let \( \psi_i : D \rightarrow R_1 \) be the projection function of \( -- i.e., \text{if } \psi(x) = z \text{ then } \psi_i(x) = z_1 \).

Let \( G \) be a uniform graph and \( \psi \) be some program with domain \( D \) and range \( R \). Let \( D \) and \( R \) be subsets of the cartesian product of \( k \) sets. Let \( L_G = \{ l_1, l_2, \ldots, l_k \} \).

**Definition 2.2-2:** A program \( \psi \) is transformed to \( G \) by a set \( TR = \{ TR_1, TR_2, TR_3, TR_4 \} \) of one-one functions where:

1. \( TR_1 : D \rightarrow V_G \).

2. \( TR_2 : (D_1 \cup R_1) \cup (D_2 \cup R_2) \cup \ldots \cup (D_k \cup R_k) \rightarrow E_G \). If \( x_i \notin (D_1 \cup R_1) \) and if \( TR_2(x_i) = e_a \) then \( G_E(e_a) = l_i \).

3. \( TR_3 : D_1' \cup D_2' \cup \ldots \cup D_k' \rightarrow S_G \). If \( x_i \notin D_1' \) and if \( TR_3(x_i) = v_x \) then \( G_{IO}(v_x) = l_i \).

4. \( TR_4 : R_1' \cup R_2' \cup \ldots \cup R_k' \rightarrow S_G \). If \( x_i \notin R_1' \) and if \( TR_4(x_i) = v_x \) then \( G_{IO}(v_x) = l_i \).

5. for every \( x \) and \( z \) if \( \psi(x) = z \) and \( TR_1(x) = v_x \), \( TR_3(x_i) = e_a \) and \( TR_3(z_i) = e_b \) then \( e_a \) is directed into \( v_x \) and \( e_b \) is directed out of \( v_x \).

\( TR \) always transforms \( \psi \) into a uniform graph. For any \( i \), every element in \( Y_i \) is assigned either an edge labelled \( l_i \) or a source or sink vertex labelled \( l_i \) by the transformation functions. Henceforth we will
refer to uniform graphs as program graphs and we will denote a program
graph as G. Our program graphs are restricted versions of data-flow
graphs. Unlike in program graphs the computation vertices in data-flow
graphs represent different computable functions. In the following
example we will illustrate a program and its transformation into a
program graph.

Example 2.2-1: Consider the problem of multiplying a band matrix \( M \)
by a vector \( X \) as shown in figure 2.2-1.

\[
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
y_5 \\
y_6
\end{bmatrix} =
\begin{bmatrix}
a_{11} & a_{12} & 0 & 0 & 0 & 0 \\
a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\
a_{31} & a_{32} & a_{33} & a_{34} & 0 & 0 \\
a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & 0 \\
a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & 0 \\
a_{61} & a_{62} & a_{63} & a_{64} & a_{65}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6
\end{bmatrix}
\]

Figure 2.2-1: Band Matrix Multiplication by a Vector

The elements in the product vector \( Y \) can be computed by the following
recurrence:

\[
y_{i(k+1)} = y_{i(k)} + a_{ik} x_k
\]

This recurrence can be rewritten as:

\[
y_{i(k+1)} = y_{i(k)} + a_{i(k+1)} x_{k(1)}
\]

\(^1\)throughout this paper 'x' denotes multiplication unless explicitly
mentioned otherwise
\[ a_{ik}^{(2)} = a_{ik}^{(1)} \]
\[ x_{k}^{(i+1)} = x_{k}^{(1)} \]

Define \( Y_1 \), \( I_1 \) and \( O_1 \) as follows:

1. \( Y_1 = \{ y_{i}^{(k)} \mid 1 \leq i \leq 6, 1 \leq k \leq 4 + i \text{ for } 1 \leq i \leq 3 \text{ and } i - 2 \leq k \leq 6 \text{ for } 4 \leq i \leq 6 \} \)

2. \( I_1 = \{ y_{i}^{(1)} \text{ and } y_{j}^{(j-2)} \mid 1 \leq i \leq 3 \text{ and } 4 \leq j \leq 6 \} \). Every \( y_{i}^{(1)} = y_{j}^{(j-2)} = 0 \)

3. \( O_1 = \{ y_{i}^{(2+1)} \text{ and } y_{j}^{(6)} \mid 1 \leq i \leq 4, 5 \leq j \leq 6 \} \). For every \( y_{i}^{(2+1)} \) and \( y_{j}^{(6)} \) in \( O_1 \), \( y_{i}^{(2+1)} = y_{i} \) and \( y_{j}^{(6)} = y_{j} \)

Define \( Y_2 \), \( I_2 \) and \( O_2 \) as follows:

1. \( Y_2 = \{ a_{ik}^{(1)} \mid 1 \leq i \leq 2, 1 \leq k \leq 6, 1 \leq k \leq 1 + i \text{ for } 1 \leq i \leq 3 \text{ and } i - 2 \leq k \leq 5 \text{ for } 4 \leq i \leq 6 \} \)

2. \( I_2 = \{ a_{ik}^{(1)} \mid i \text{ and } k \text{ vary as in } Y_2 \} \). For every \( a_{ik}^{(1)} \) in \( I_2 \), \( a_{ik}^{(1)} = a_{ik} \).

3. \( O_2 = \{ a_{ik}^{(2)} \mid i \text{ and } k \text{ vary as in } Y_2 \} \).

Define \( Y_3 \), \( I_3 \) and \( O_3 \) as follows:

1. \( Y_3 = \{ x_{k}^{(1)} \mid 1 \leq k \leq 5, i \leq k \leq k + 3 \text{ for } 1 \leq k \leq 2, 2 \leq k \leq 6 \text{ for } k = 3 \text{ and } k - 1 \leq i \leq 7 \text{ for } 4 \leq k \leq 5 \} \).

2. \( I_3 = \{ x_{k}^{(1)} \text{ and } x_{j}^{(j-1)} \mid 1 \leq k \leq 2, 3 \leq j \leq 5 \} \). For every \( x_{k}^{(1)} \) and
\[ x_j^{(j-1)} \text{ in } I_3, \ x_k^{(1)} = x_k \text{ and } x_j^{(j-1)} = x_j. \]

3. \( O_3 = \{ x_k^{(3+k)} \text{ and } x_j^{(7)} \mid 1 \leq k \leq 4 \} \).

Define \( \Psi, \Psi_1, \Psi_2 \) and \( \Psi_3 \) as follows:

1. \( \Psi(<y_i^{(k)}, a_{ik}^{(1)}, x_k^{(1)})> = <y_i^{(k+1)}, a_{ik}^{(2)}, x_k^{(i+1)}> \)

2. \( \Psi_1(<y_i^{(k)}, a_{ik}^{(1)}, x_k^{(1)})> = y_i^{(k+1)} = y_i^{(k)} + a_{ik}^{(1)} \times x_k^{(i)} \)

3. \( \Psi_2(<y_i^{(k)}, a_{ik}^{(1)}, x_k^{(1)})> = a_{ik}^{(2)} = a_{ik}^{(1)} \)

4. \( \Psi_3(<y_i^{(k)}, a_{ik}^{(1)}, x_k^{(1)})> = x_k^{(i+1)} = x_k^{(i)} \)

Define \( D = \{ <y_i^{(k)}, a_{ik}^{(1)}, x_k^{(i)}> \} \) where the tuple exists iff \( y_i^{(k)}, a_{ik}^{(1)}, x_k^{(i)} \) are all defined in \( Y_1, Y_2 \) and \( Y_3 \) respectively.

Now \( Y_1 = D_1 \cup R_1, \ Y_2 = D_2 \cup R_2, \ Y_3 = D_3 \cup R_3 \). Also \( D_1 = I_1, \ R_1 = O_1, \ D_2 = I_2, \ R_2 = O_2, \ D_3 = I_3 \) and \( R_3 = O_3 \).

Define a uniform graph \( G \) as follows:

\( L_G = \{11, 12, 13\} \).

\( V_G = \{ v_{ik} \mid 1 \leq i \leq 6, \ 1 \leq k \leq i+1 \text{ for } 1 \leq i \leq 3 \text{ and } i-2 \leq k \leq 5 \text{ for } 4 \leq i \leq 6 \} \).

\( E_G = E_{Y_1} \cup E_{Y_2} \cup E_{Y_3} \) where:

1. \( E_{Y_1} = \{ ey_{ik}^{(k)} \mid i \text{ and } k \text{ vary as in } Y_1 \} \) and for every \( ey_{ik}^{(k)} \), \( G_B(ey_{ik}^{(k)}) = 11 \).
2. \( EY_2 = \{ e_{i k}^{(j)} \mid i, k \text{ and } j \text{ vary as in } Y_2 \} \) and for every 
\( e_{i k}^{(j)} \), \( G_E(e_{i k}^{(j)}) = 12 \).

3. \( EY_3 = \{ e_{i k}^{(l)} \mid i \text{ and } k \text{ vary as in } Y_3 \} \) and for every 
\( e_{i k}^{(l)} \), \( G_E(e_{i k}^{(l)}) = 13 \).

\( S_O = S_{O1} \cup S_{O2} \cup S_{O3} \) where:

1. \( S_{O1} = \{ v_{i j}^{(1)} \text{ and } v_{j i}^{(j-2)} \mid i \text{ and } j \text{ vary as in } I_1 \} \) and for 
every \( v_{i j}^{(1)} \) and \( v_{j i}^{(j-2)} \), \( G_{I0}(v_{i j}^{(1)}) = G_{I0}(v_{j i}^{(j-2)}) = 11 \).

2. \( S_{O2} = \{ v_{i k}^{(1)} \mid i \text{ and } k \text{ vary as in } I_2 \} \) and for every 
\( v_{i k}^{(1)} \), \( G_{I0}(v_{i k}^{(1)}) = 12 \).

3. \( S_{O3} = \{ v_{i j}^{(1)} \text{ and } v_{j i}^{(i-1)} \mid k \text{ and } j \text{ vary as in } I_3 \} \) and for 
every \( v_{i j}^{(1)} \) and \( v_{j i}^{(i-1)} \), \( G_{I0}(v_{i j}^{(1)}) = G_{I0}(v_{j i}^{(i-1)}) = 13 \).

\( S_I = S_{I1} \cup S_{I2} \cup S_{I3} \) where:

1. \( S_{I1} = \{ v_{i j}^{(2+1)} \text{ and } v_{j i}^{(6)} \mid i \text{ and } j \text{ vary as in } O_1 \} \) and for 
every \( v_{i j}^{(2+1)} \) and \( v_{j i}^{(6)} \), \( G_{I0}(v_{i j}^{(2+1)}) = G_{I0}(v_{j i}^{(6)}) = 11 \).

2. \( S_{I2} = \{ v_{i k}^{(2)} \mid i \text{ and } k \text{ vary as in } O_2 \} \) and for every 
\( v_{i k}^{(2)} \), \( G_{I0}(v_{i k}^{(2)}) = 12 \).

3. \( S_{I3} = \{ v_{i j}^{(3+k)} \text{ and } v_{j i}^{(7)} \mid k \text{ varies as in } O_3 \} \) and for every 
\( v_{i j}^{(3+k)} \), \( G_{I0}(v_{i j}^{(3+k)}) = G_{I0}(v_{j i}^{(7)}) = 13 \).

Define \( TR_1 \), \( TR_2 \), \( TR_3 \) and \( TR_4 \) as follows:
1. \( \text{TR}_1(\langle y_1^{(k)}, a_{1k}^{(1)}, x_k^{(1)} \rangle) = v_{1k} \)

2. \( \text{TR}_2(y_1^{(k)}) = e y_1^{(k)}, \text{TR}_2(a_{1k}^{(j)}) = e a_{1k}^{(j)} \) and \( \text{TR}_2(x_k^{(1)}) = e x_k^{(1)} \).

3. \( \text{TR}_3(y_1^{(1)}) = v y_1^{(1)} \) and \( \text{TR}_3(y_j^{(j-2)}) = v y_j^{(j-2)} \); \( \text{TR}_3(a_{1k}^{(1)}) = v a_{1k}^{(1)} \);
   \( \text{TR}_3(x_k^{(1)}) = v x_k^{(1)} \) and \( \text{TR}_3(x_j^{(j-1)}) = v x_j^{(j-1)} \).

4. \( \text{TR}_4(y_1^{(2+1)}) = v y_1^{(2+1)} \) and \( \text{TR}_4(y_j^{(6)}) = v y_j^{(6)} \); \( \text{TR}_4(a_{1k}^{(2)}) = v a_{1k}^{(2)} \);
   \( \text{TR}_4(x_k^{(3+k)}) = v x_k^{(3+k)} \) and \( \text{TR}_4(x_5^{(7)}) = v x_5^{(7)} \).

The resulting program graph is shown in figure 2.2-2.

![Figure 2.2-2](image)

In figure 2.2-2 'o' represents computation vertices and '∅' represents source or sink vertices. The horizontal, vertical and oblique edges are labelled 11, 13 and 12 respectively. If a source or sink vertex is connected to a computation vertex by a horizontal, vertical or oblique edge then that source or sink vertex is labelled 11, 13 and 12 respectively. A computation vertex in figure figure 2.2-2 is shown in
greater detail in figure 2.2-3.

![Diagram](image)

Figure 2.2-3

We will be often referring to the contents or the element represented by an edge or a source vertex or a sink vertex with the explicit understanding that the contents or the element represented refer to the element in some domain — say $Y_j$ — that has been transformed to either an edge or a source vertex or a sink vertex by the transformation functions.

2.3. Processor and Linear-Array Models

A general-purpose processor model consists of an addressable-memory, a control-unit and an instruction-set. During a cycle the processor executes an instruction from its instruction-set where a cycle is the total time taken starting from the time to fetch an instruction to completion of the instruction. In every cycle the processor fetches an instruction from memory, decodes the instruction, fetches the operands from memory, executes the instruction and stores the results in memory.

Obviously the processor needs decision-making ability to:
1. execute different instructions in different cycles

2. to access different operand addresses in memory during instruction execution

This decision-making ability is achieved by having states and making state-transitions. In every cycle the control-unit of a processor receives control-inputs (instruction and operand addresses) and determines on the basis of the control-inputs the state the processor should be in.

In this paper we will be using a very simple model of a processor. Our processor-model does not have any decision-making ability. The implications are:

1. the processor must execute the same instruction in every cycle

2. the operand addresses for the instruction is the same in every cycle

We give a formal definition of a processor in the following:

Definition 2.3-1: A processor is a 8-tuple \( P = \langle I_P, O_P, D_P, L_P, F_I, F_O, F_D, W_F \rangle \) where:

1. \( I_P \) is a set of input ports

2. \( O_P \) is a set of output ports
3. $D_P$ is a set of non-zero positive integers

4. $L_P$ is a set of labels

5. $F_I$, $F_O$ and $F_D$ are three one-one functions such that:
   a. $F_I : I_P \rightarrow L_P$
   b. $F_O : O_P \rightarrow L_P$
   c. $F_D : D_P \rightarrow L_P$

6. $\Psi_P$ is a $|L_P|$-ary function computed by the processor in every cycle, i.e., if $IN_C$ is the input to $P$ in cycle $c$ and $OUT_C$ is the output computed by $P$ in cycle $c$ then $\Psi_P(IN_C) = OUT_C$ and $IN_C$ and $OUT_C$ are both $|L_P|$-tuples.

Let $L_P=\{11, 12, \ldots, 1k\}$ for a processor $P$. If $x \in D_P$ and $F_D(x)=lj$ then we will denote $x$ as $d_{lj}$ and refer to it as the delay associated with label $lj$. Now $|L_P|=k$. Every input-port in $I_P$ is assigned a unique label from $L_P$. Similarly every output-port in $O_P$ is also assigned a unique label from $L_P$ and so $|I_P|=|O_P|=k$. The function $\Psi_P$ is the instruction executed by $P$ in every cycle. $\Psi_P$ is a 'simple' function, i.e., it can be computed using a 'few' instructions from the instruction-set of a machine like IBM/360.

Let the $k$-tuple $IN_C$ be the input to processor $P$ in cycle $c$. The processor computes the $k$-tuple $OUT_C \Psi_P(IN_C)$ in cycle $c$. $\Psi_P$ can be
decomposed as the cross-product of \( k \) functions, i.e.,
\[
\text{OUT}_c = \psi_1(\text{IN}_c) \times \psi_2(\text{IN}_c) \times \ldots \times \psi_k(\text{IN}_c).
\]
Now each component of \( \text{IN}_c \) and \( \text{OUT}_c \) is a scalar value, i.e., a value typically held in a memory location like IBM/360. Processor \( P \) receives every component of \( \text{IN}_c \) from a distinct input-port and outputs every component of \( \text{OUT}_c \) onto a distinct output-port. Let \( \text{IN}_c = \langle \text{in}_{11}, \text{in}_{12}, \ldots, \text{in}_{1k} \rangle \) and \( \text{OUT}_c = \langle \text{out}_{11}, \text{out}_{12}, \ldots, \text{out}_{1k} \rangle \). Now \( \text{in}_{1j} \) is the \( j \)th component of \( \text{IN}_c \) that is received by the processor at the input-port labelled \( lj \) and \( \text{out}_{1j} \) is the \( j \)th component of \( \text{OUT}_c \) that the processor outputs on the output-port labelled \( lj \) after a delay of \( d_{lj} \) cycles, i.e., in cycle \( c+d_{lj} \). Also \( \text{out}_{1j} = \psi_j(\text{IN}_c) \). However in every cycle between \( c \) and \( c+d_{lj} \) a new output-tuple is computed by the processor. Now each of these output-tuples is comprised of a component labelled \( lj \) (\( \text{out}_{lj} \)) and so the number of such components produced between \( c \) and \( c+d_{lj} \) is \( d_{lj} \). Each of these output components are outputted from the output-port labelled \( lj \) only after cycle \( c+d_{lj} \) and so all these values need to be buffered in a queue. For every label \( lj \) such a queue is implemented by a shift-register of length \( d_{lj} - 1 \). The rightmost end (output of shift-register) of the shift-register is connected to the output port labelled \( lj \). During every cycle \( c \) processor \( P \) does the following:

1. compute \( \text{OUT}_c = \psi_p(\text{IN}_c) \)

2. for all \( lj \), shift the contents of the shift-register to the right by one position. This creates a vacant position at the leftmost end (input of shift-register). Place \( \text{out}_{lj} \) in this vacant slot.
Example 2.3-1:

![Diagram of a Processor]

Figure 2.3-1: A Processor

In figure 2.3-1 $L_p=\{11, 12, 13\}$. $d_{11}=4$, $d_{12}=5$ and $d_{13}=1$.

Definition 2.3-2: Two processors are identical iff:

1. the label-sets are the same

2. the functions computed are identical

3. identical labels in both the processors have the same delays

Let PR be a totally ordered set of identical processors. The processors are numbered $\{\emptyset, 1, \ldots, |PR|-1\}$. We will refer to a processor in this set by its index number in the ordering. For every label $l_j$ let $B_{1j}$ be a binary relation on PR.

Definition 2.3-3: For any pair of processors $m$ and $n$ in PR, $m \sim B_{1j} n$ iff the output port labelled $l_j$ of $m$ is connected to the input port labelled $l_j$ of $n$.

$B_{1j}$ is a neighbourhood relation on processors. If two processors are
related by $B_{lj}$ then they are neighbours with respect to the label $lj$.

**Definition 2.3-4:** $B_{lj}$ is empty iff there exists no pair $m$ and $n$ in PR such that $m B_{lj} n$ and is non-empty otherwise.

**Definition 2.3-5:** A linear-array of interconnected processors is a 4-tuple $LA=<PR, L_P, \Psi_P, N_P>$ where:

1. PR is a totally ordered set of identical processors

2. $L_P$ is the label-set of every processor in PR

3. $\Psi_P$ is the function computed by every processor in PR

4. $N_P=\{n_{lj}\}$ iff $B_{lj}$ is non-empty is a set of constants such that every $n_{lj}$ is only one of $\{+1, -1, \emptyset\}$.

If $n_{lj}=1$ then it means that for all processors $m$ its neighbour with respect to label $lj$ is processor $m+1$.

If $n_{lj}=-1$ then it means that for all processors $m$ its neighbour with respect to label $lj$ is processor $m-1$.

If $n_{lj}=\emptyset$ then it means that for all processors $m$ its neighbour with respect to label $lj$ is processor $m$ itself.

The linear array of processors is driven by a single-phase global clock. In every clock pulse all the processors compute $\Psi_P$. The cycle time for a processor is the width of the clock.
The input-ports and output-ports of some processors are designated to interact with the external environment. Communication with the external environment occurs only through these ports. These input-ports and output-ports are the distinguished input-ports and output-ports respectively. Data is driven into the array only through the distinguished input-ports and the results are obtained through the distinguished output-ports. For every label 1j this designation depends on $B_{1j}$, namely:

1. if $B_{1j}$ is empty the input-port and output-port labelled 1j of every processor is the distinguished input-port and output-port respectively.

2. if $n_{1j} = 1$ then the input-port labelled 1j of processor numbered $\emptyset$ and the output-port labelled 1j of processor numbered $|PR| - 1$ are the distinguished input-ports and output-ports respectively.

3. if $n_{1j} = -1$ the the input-port labelled 1j of processor numbered $|PR| - 1$ and the output-port of processor numbered $\emptyset$ are the distinguished input-port and output-port respectively.

4. if $n_{1j} = \emptyset$ then there are no distinguished input-ports and output-ports labelled 1j.

Example 2.3-2:
Figure 2.3-2: Linear Array of Processors

In figure 2.3-2 |PR|=4 and \( L_P=\{11, 12, 13, 14\} \). \( B_{14} \) is empty and so \( N_P=\{n_{11}, n_{12}, n_{13}\} \) and \( n_{11}=1, n_{12}=1 \) and \( n_{13}=\emptyset \). \( I_{11} \) and \( O_{11} \) are the distinguished input-ports and output-ports for 11. \( I_{12} \) and \( O_{12} \) are the distinguished input-ports and output-ports for 12. \( I_{14} \) and \( O_{14} \) are the distinguished input-ports and output-ports for 14. 13 does not have any distinguished input-ports and output-ports. A simple register serves as the input/output port labelled 13.

3. Mapping Programs on Linear-Arrays

In this section we formally define the problem of mapping programs on linear arrays. We first provide an intuitive view of mapping.

Consider a program\(^1\) whose domain \( D \) and range \( R \) are subsets of \( Y_1 \times Y_2 \times \cdots \times Y_k \). Let \( G \) be the corresponding program graph of \( \Psi \) which is obtained by transforming \( \Psi \) into \( G \) using the set \( TR=\{TR_1, TR_2, TR_3, TR_4\} \) of transformation functions. Let \( L_G=\{11, 12, \ldots, 1k\} \). In particular consider a path in \( G \) as shown in figure 3.1-1 and whose path-label is

\(^1\)we use the same notations as in section-2
Figure 3.1-1: Path labelled \( l_j \)

In figure 3.1-1, \( v_1, v_2, \ldots, v_m \) are computation vertices. \( v_s \) and \( v_f \) are source and sink vertices respectively and are labelled \( l_j \). \( e_0, e_1, \ldots, e_m \) are edges whose labels are \( l_j \).

Let \( z_0, z_1, \ldots, z_{(m-1)} \) and \( w_1, w_2, \ldots, w_m \) be tuples in \( Y \) such that for every \( i \), \( 1 \leq i \leq m \), \( (z_{(i-1)})=w_i \) and \( TR_1(z_{(i-1)})=v_i \). Let \( x_{(i-1)} \) denote the \( j \)th component of \( z_{(i-1)} \) and hence \( TR_2(x_{(i-1)})=e_{(i-1)} \). Also let \( TR_3(x_0)=v_s \) and \( TR_4(x_m)=v_f \).

Let \( z_i(-j) \) denote \( z_i \) without its \( j \)th component, i.e., if \( z_i=<u_1, u_2, \ldots, u_{j-1}, x_i, u_{j+1}, \ldots, u_k> \) then \( z_i(-j)=<u_1, u_2, \ldots, u_{j-1}, u_{j+1}, \ldots, u_k> \).

Now \( x_{i+1} = \psi_j(z_i(-j), x_i) \)

\[ = \psi_j(z_i(-j), \psi_j(z_{i-1}, \psi_j(\ldots, \psi_j(z_0(-j), x_0)) \ldots)). \]

So \( x_{i+1} \) is the result of an \( i \)-fold composition of \( \psi_j \) on the sequence of arguments \( z_0, z_1, \ldots, z_i \). In a sequential processing of the program \( \psi \) a register is associated with every labelled path. In particular for
the path in figure 3.1-1 the location is initialized with the element represented by the source vertex \( v_s \) which is \( x_0 \) and at the end of the computation the location contains the element represented by the sink vertex \( v_f \) which is \( x_m \). If any vertex \( v_i \) is computed then the element represented by edge \( e_i \) which is \( x_i \) is stored in the location and is held there till the computation of \( v_{i+1} \) for which the tuple \( z_i \) is the input and \( x_i \) is the \( j \)th component of \( z_i \).

A mapping of \( \Psi \) on a linear-array of processors is an execution of every computation vertex in \( G \) by some processor in the array. However this execution is performed under some constraints.

Let \( LA \) be a linear-array of processors with \( L_p = \{l_1, l_2, \ldots, l_k\} \). Let \( \Psi \) and \( G \) be defined as in the beginning of this section. Let \( T = \{0, c, 2c, \ldots\} \) be a sequence of time steps where \( c \) is a basic processor cycle time. Let \( S^G \) and \( S^G_t \) be a subset of \( (S^G)x(T) \) and \( (S^G_t)x(T) \) respectively.

Definition 3.0-1: A mapping of \( G \) onto the processors in \( LA \) is a 4-tuple \( MP_G = \langle TA, PA, IOA \rangle \) where \( TA, PA \) and \( IOA \) are many-one functions such that:

1. \( TA: V_G \rightarrow T \)

2. \( PA: V_G \rightarrow N \)

3. \( IOA: (S^G \cup S^G_t) \rightarrow N \)

If \( v_x \) is a source vertex such that \( IOA(v_x, t) = s \) then it means that \( v_x \)
is mapped onto the input-port of s at time t and if \( v_x \) is a sink vertex such that \( \text{IOA}(v_x, t) = s \) then it means that \( v_y \) is mapped onto the output-port of s at time t. \( S_G', S_I', \) and \( \text{IOA} \) are constructed as follows:

1. Initially \( S_G' \) and \( S_I' \) are empty.

2. For all \( lj \) if \( v_x \) is in \( S_G \) and there exists an edge labelled \( lj \) directed from \( v_x \) to \( v_y \) then \( S_G' = S_G' \cup \{ \langle v_x, t \rangle \} \) where \( t = \text{TA}(v_y) - m \cdot d_{lj} \) for all \( m \) such that \( \text{PA}(v_y) - m \cdot n_{lj} \) is a processor index. Define \( \text{IOA}(v_x, t) = \text{PA}(v_x) - m \cdot n_{lj} \)

3. For all \( lj \) if \( v_x \) is in \( S_I \) and there exists an edge labelled \( lj \) directed from \( v_x \) to \( v_y \) then \( S_I' = S_I' \cup \{ \langle v_x, t+d_{lj} \rangle \} \) where \( t = \text{TA}(v_y) + m \cdot n_{lj} \) for all \( m \) such that \( \text{PA}(v_y) + m \cdot n_{lj} \) is a processor index. Define \( \text{IOA}(v_x, t+d_{lj}) = \text{PA}(v_x) + m \cdot n_{lj} \)

Let \( t_s \) and \( t_f \) denote the start time and finish time respectively of a computation graph \( G \) that has been mapped onto \( LA \).

\( t_s \) is less or equal to the minimum over all \( t \) in all tuples \( \langle v_x, t \rangle \) in \( S_G' \) and for all labels \( lj \) such that \( G_{IO}(v_x) = lj \) and \( n_{lj} \neq 0 \).

\( t_f \) is greater or equal to the maximum over all \( t \) in all tuples \( \langle v_x, t \rangle \) in \( S_I' \) and for all labels \( lj \) such that \( G_{IO}(v_x) = lj \) and \( n_{lj} \neq 0 \).

Definition 3.0-2: \( MP_G \) is syntactically correct iff:

1. \( \Psi = \Psi_P \).
2. the three functions satisfy the following:

a. For every label \( l_j \) if \( v_x \) and \( v_y \) are computation vertices and there exists an edge labelled \( l_j \) directed from \( v_x \) to \( v_y \) then \( PA(v_y) = PA(v_x) + n_{l_j} \) and \( TA(v_y) = TA(v_x) + d_{l_j} \).

b. If \( v_x \) and \( v_y \) are computation vertices and if \( PA(v_x) = PA(v_y) \) and \( TA(v_x) = TA(v_y) \) then \( v_x = v_y \).

c. If \( v_x \) and \( v_y \) are either source or sink vertices and if \( IOA(v_x, t) = IOA(v_y, t) \) and \( G_{IO}(v_x) = G_{IO}(v_y) \) then \( v_x = v_y \).

3. for all \( l_j \) if \( n_{l_j} \neq 0 \) and \( PA(v_x) = PA(v_y) \) then there must be a major path labelled \( l_j \) that passes through these two vertices (i.e., a register is associated with the computation of the vertices in the path labelled \( l_j \) and this register serves as the input/output port labelled \( l_j \) for the processor).

For any tuple \( <v_x, t_m> \) either in \( SI_G \) or in \( SO_G \), let \( t_m(v_x) \) denote the contents of \( v_x \) at time \( t_m \).

**Definition 3.0-3:** \( MP_G \) correctly computes \( \Psi \) iff:

1. \( MP_G \) is syntactically correct

2. For any pair of tuples \( <v_x, t_m> \) and \( <v_x, t_n> \) either in \( SI_G \) or in \( SO_G \), \( t_m(v_x) = t_n(v_x) \)
Intuitively if there exists more than one tuple $<v_x, t>$ either in $SOG'$ or in $SIG'$ it means that $v_x$ is mapped onto the input/output ports of two distinct processors more than once. Hence the contents of the source and sink vertices that is mapped more than once must be invariant after every such mapping.

4. Preliminary Results

Let $G$ be the program graph for some program $P$. Let the label set $L_G = \{l_1, l_2, \ldots, l_k\}$. Let LA be the linear array of processors whose label set $L_P$ is the same as $L_G$ and the function $\Psi_P$ computed by any processor in LA is the same as $\Psi$.

For any label $l_j$ in $L_G$ let $E_{l_j}$ be the set of major paths labelled $l_j$.

Definition 4.0-1: For any label $l_i$ and $l_j$, $E_{l_i}$ is identical to $E_{l_j}$ iff for every major path in $E_{l_i}$ there is an identical path in $E_{l_j}$ and vice-versa.

Let $H = \{E_{l_j} | r_{l_j} \text{ is not empty} \}$ and for any $E_{l_j}$ and $E_{l_i}$ in $H$, $E_{l_j}$ is not identical to $E_{l_i}$.

Let $F = \{E_{l_j} | r_{l_j} \text{ is not empty and for every } E_{l_j} \text{ in } F \text{ there exists some } E_{l_i} \text{ in } H \text{ such that } E_{l_j} \text{ and } E_{l_i} \text{ are identical} \}$.

Let $D = \{E_{l_j} | r_{l_j} \text{ is empty} \}$.

---

$^1$Definition 2.1-6, page 7
Let q be a undirected\(^2\) path from a computation vertex \(v_x\) to another computation vertex \(v_y\) such that all the vertices in the path are computation vertices and all the edges in the path have the same label. Let \(l_j\) be the label of the edges in q. A directed edge imposes an order on the pair of vertices it connects. If there is a directed edge from \(v_x\) to \(v_y\) then the order of \(v_x\) is less than the order of \(v_y\). A path imposes an order on the vertices it visits. If the path visits \(v_x\) before \(v_y\) then the order of \(v_x\) is less than the order of \(v_y\). In a directed path the order of vertices imposed by the path is consistent with the order imposed by the edges. In a undirected path there could be edges in the path such that the order imposed by these edges on the vertices is not consistent with the order imposed by the path. Let \(k_1\) and \(k_2\) be the number of edges in a undirected path that impose an order that is consistent and not consistent respectively with the order imposed by the undirected path. If \(v_x\) and \(v_y\) are two computation vertices in this path then the processors computing \(v_x\) and \(v_y\) along with the time at which they are computed can be easily related.

Proposition 4.0-1: In any syntactically correct mapping

\[ PA(v_y) = PA(v_x) + (k_1 - k_2) * n_{1j} \] and \[ TA(v_y) = TA(v_x) + (k_1 - k_2) * d_{1j} \].

Let q be a major path labelled \(l_j\). Let \(v_x\) and \(v_y\) be two computation vertices in q such that the distance from \(v_x\) to \(v_y\) is \(k\).

Corollary 4.0-1: In any syntactically correct mapping,

\[ \text{in a undirected path the direction of edges in the path are ignored} \]
PA(v_y) = PA(v_x) + k * n_j and TA(v_y) = TA(v_x) + k * d_j.

Proposition 4.0–2: If a major path has k computation vertices then the indices of these computation vertices in the ordering imposed by a topological sort on the vertices in the major path range from 1 to k.

Proof: A topological sort on a directed path from v_x to v_y containing m vertices in the path (v_x and v_y included) imposes an ordering on these vertices and assigns indices to the vertices in the ordering ranging from 0 to m–1. Index 0 is assigned to v_x and index m–1 is assigned to v_y and indices ranging from 1 to m–2 are assigned to all the m–2 intermediate vertices in the path (v_x and v_y included).

In any major path the source vertex is v_x, the sink vertex is v_y and the intermediate vertices are computation vertices.

Proposition 4.0–3: G must be acyclic for any syntactically correct mapping.

Proof: Suppose G has a cycle. Then there must be a computation vertex v_x such that there is a directed path from v_x to itself and hence in any syntactically correct mapping TA(v_x) > TA(v_y).

Proposition 4.0–4:

If there are two major paths in G such that the computation vertices in these two major paths are same then these two major paths must be identical for any syntactically correct mapping.
Proof: Let \( q_r \) and \( q_s \) be the two major paths such that the computation vertices in these two major paths are the same. Let \( k \) be the number of computation vertices in \( q_r \) and \( q_s \). The range of the indices of the computation vertices (i.e., 1 to \( k \)) in the ordering imposed by a topological sort on \( q_r \) is the same as the range of the indices of the computation vertices in the ordering imposed by a topological sort on \( q_s \).

Suppose \( q_r \) is not identical to \( q_s \). It can be easily shown that there must exist two computation vertices \( v_x \) and \( v_y \) such that \( q_r \) visits \( v_x \) before \( v_y \) and \( q_s \) visits \( v_y \) before \( v_x \) and hence in any syntactically correct mapping \( TA(v_x) > TA(v_y) \) in \( q_s \) and \( TA(v_y) > TA(v_x) \) in \( q_r \).

Proposition 4.0-5:

For any \( 1j \) in \( L_G \) if \( r_{1j} \) is not empty then in any syntactically correct mapping \( n_{1j} \) must be one of \( \{1, -1, \emptyset\} \).

Lemma 4.0-1:

For any \( 1i \) and \( 1j \) in \( L_G \) if \( E_{1i} \) and \( E_{1j} \) are in \( H \) then in any syntactically correct mapping \( n_{1i} = n_{1j} \neq \emptyset \).

Proof: If \( E_{1i} \) and \( E_{1j} \) are in \( E \) then \( E_{1i} \) and \( E_{1j} \) are not identical. Consequently there is a major path \( q_r \) in \( E_{1i} \) that is not identical to any major path in \( E_{1j} \) and this is because of one of the following:

1. there is a major path \( q_s \) in \( E_{1j} \) such that the computation
vertices in \( q_s \) and in \( q_r \) are the same but there exists a computation vertex \( v_x \) such that its index in the ordering imposed by a topological sort on \( q_r \) is different from its index in the ordering imposed by a topological sort on \( q_s \). But by proposition 4.0-4 \( q_r \) and \( q_s \) must be identical.

2. there does not exist any major path \( q_s \) in \( E_{1j} \) such that the computation vertices in \( q_r \) and \( q_s \) are the same and hence,

a. there exists more than one major path in \( E_{1j} \) and each of these major paths pass through a subset of the computation vertices in \( q_r \).

b. there exists a major path \( q_s \) in \( E_{1j} \) and a subset of the computation vertices in \( q_s \) is the same as the computation vertices in \( q_r \).

First let us consider (2a). Without loss of generality let there be two major paths \( q_s \) and \( q_t \) in \( E_{1j} \) that pass through computation vertices in \( q_r \) as shown in figure 4.0-1.

Figure 4.0-1
If \( n_{1i} = n_{1j} = 0 \) then \( PA(v_x) = PA(v_y) \). By definition of a syntactically correct mapping a single major path labelled \( lj \) must pass through \( v_x \) and \( v_y \). But in figure 4.0-1 \( v_x \) and \( v_y \) are on distinct major paths \( q_s \) and \( q_t \) respectively.

Next consider (2b). Let \( q_s \) be the major path in \( E_{1j} \) that passes through all the computation vertices in \( q_r \) as shown in figure 4.0-2.

![Figure 4.0-2](image)

The major path \( q_r \) is directed from \( v_x \) to \( v_y \) through edges shown below the dashed line. \( q_s \) is the major path that visits \( v_x \), \( v_y \) and \( v_z \) in that order. All the vertices in the path of \( q_s \) from \( v_x \) to \( v_y \) is the same as the vertices in \( q_r \). The edges in the path of \( q_s \) from \( v_x \) to \( v_y \) are through edges shown above the dashed line. Now \( v_z \) is a computation vertex in the path of \( q_s \) which is also in the path of another major path \( q_t \) that is labelled \( li \). Now \( q_t \) is distinct from \( q_r \). So if \( n_{1i} = n_{1j} = 0 \) then \( PA(v_x) = PA(v_y) = PA(v_z) \). But in a syntactically correct mapping there must be a single major path labelled \( li \) which passes through \( v_y \) and \( v_z \). But in figure 4.0-2 \( v_y \) and \( v_z \) are in distinct major paths \( q_r \) and \( q_t \) respectively.
Proposition 4.0–6:

In any syntactically correct mapping if \( v_u \) and \( v_w \) are the source and sink vertices of a major path then \( \text{IOA}\langle v_u, t \rangle = \text{IOA}\langle v_w, t \rangle \).

Proof: There must be at least one computation vertex in between the source and sink vertex and hence the result.

Let \( T_{<a,b>} \) be a binary relation on computation vertices that have been mapped onto processors by some mapping \( MP_G \).

Definition 4.0–2: \( v_x \ T_{<a,b>} \ v_y \) iff

1. \( PA(v_y) = PA(v_x) + a \) and \( PA(v_y) \) is a processor index.

2. \( TA(v_y) = TA(v_x) + b \).

So \( T_{<a,b>}^k \) is the \( k \)-fold composition of \( T_{<a,b>} \) and hence if \( v_x \ T_{<a,b>}^k \ v_y \) then \( PA(v_y) = PA(v_x) + k \cdot a \) and \( TA(v_y) = TA(v_x) + k \cdot b \).

Now consider a mapping \( M \) that satisfies all the properties of a syntactically correct mapping except (2c). The conditions under which \( M \) satisfies (2c) is stated in the following two lemmas.

Lemma 4.0–2:

For any label \( lj \) such that \( n_{lj} = \emptyset \) if \( G_{IO}(v_u) = G_{IO}(v_w) = lj \) for any \( v_u, v_w \) in \( SI_G \cup SI_G \) and \( \text{IOA}\langle v_u, t \rangle = \text{IOA}\langle v_w, t \rangle \) in \( M \) then \( v_u = v_w \).

Proof: Suppose the hypothesis is true but \( v_u \neq v_w \). Then by proposition 4.0–6 \( v_u \) and \( v_w \) must be in distinct major paths.
labelled \( l_j \). This means there exists a computation vertex \( v_x \) in the major path in which \( v_u \) is a source vertex and another computation vertex \( v_y \) in the major path in which \( v_w \) is a source vertex respectively. \( n_{l_j} = \emptyset \) and hence \( PA(v_x) = PA(v_y) \). But this violates property of a syntactically correct mapping and we have assumed that \( M \) satisfies this property.

**Lemma 4.0-3:**

For any label \( l_j \) such that \( n_{l_j} \neq \emptyset \) if \( G_{IO}(v_u) = G_{IO}(v_w) = l_j \) for any \( v_u \) and \( v_w \) in \( SI_G \cup SO_G \) and \( IOA(v_u, t) \neq IOA(v_w, t) \) in \( M \) then \( v_u = v_w \) iff for any pair of computation vertices \( v_x \) and \( v_y \) if \( v_x \nearrow^{k_{l_j}} v_y \) where \( a = n_{l_j} \) and \( b = d_{l_j} \) then there must be a major path labelled \( l_j \) passing from \( v_x \) to \( v_y \) in that order and the distance from \( v_x \) to \( v_y \) in this path must be \( k \).

**Proof:**

**Necessity:**

Suppose there exist computation vertices \( v_x \) and \( v_y \) and a label \( l_j \) such that \( v_x \nearrow^{k_{l_j}} v_y \) in \( M \) and there does not exist a single major path labelled \( l_j \) passing through \( v_x \) and \( v_y \). Let \( q_r \) and \( q_s \) be two distinct major paths labelled \( l_j \) such that \( v_x \) is in \( q_r \) and \( v_y \) is in \( q_s \) respectively. Let \( v_u \) and \( v_w \) be the source vertices for \( q_r \) and \( q_s \) respectively. Let \( k_1 \) and \( k_2 \) be the distances from \( v_u \) to \( v_x \) and from \( v_w \) to \( v_y \) respectively. Let \( PA(v_x) = s \) and \( TA(v_x) = t \). Consequently \( v_u \) is mapped onto the input port of processor \( s - k_1 \cdot n_{l_j} \) at \( t - k_1 \cdot d_{l_j} \). Similarly \( v_w \) is mapped onto the input port of \( s + (k - k_2) \cdot n_{l_j} \) at \( t + (k - k_2) \cdot d_{l_j} \). Without
loss of generality let \((s-k_1) \cdot n_{1j} + (s+(k-k_2) \cdot n_{1j})\). Since \(n_{1j}\) is one of \(\{1,-1\}\) there must exist a \(k_3\) such that \(s-k_1 \cdot n_{1j} = s+(k-k_2-k_3) \cdot n_{1j}\) and so \(k_1 = -(k-k_2-k_3)\). Now by our construction of \(SIC'\) there must exist a tuple \(\langle v_w, t+(k-k_2-k_3) \cdot d_{1j} \rangle\) in \(SIC'\) such that \(IOA\langle v_w, t+(k-k_2-k_3) \cdot d_{1j} \rangle = s+(k-k_2-k_3) \cdot n_{1j} = s-k_1 \cdot n_{1j}\) and hence \(IOA\langle v_u, t-k_1 \cdot d_{1j} \rangle = IOA\langle v_w, t-k_1 \cdot d_{1j} \rangle\) and \(v_u = v_w\).

**Sufficiency**

Suppose there exist vertices \(v_u\) and \(v_w\) in \(SIC\) \(SO_G\) and a label \(1j\) such that \(IOA\langle v_u, t \rangle = IOA\langle v_w, t \rangle = s\) in \(M\), \(G_{IO}(v_u) = G_{IO}(v_w) = 1j\) and \(v_u \not= v_w\). By proposition 4.0-6 \(v_u\) and \(v_w\) must be in two distinct major paths labelled \(1j\). \(v_u\) and \(v_w\) in combination must be one of the following:

1. both are source vertices
2. both are sink vertices
3. one is a source vertex and the other is a sink vertex

Consider the first case where \(v_u\) and \(v_w\) are both source vertices. By lemma 4.0-2 \(n_{1j} \not= 0\). Let \(q_r\) and \(q_s\) be the two major paths labelled \(1j\) such that \(v_u\) is the source in \(q_r\) and \(v_w\) is the source in \(q_s\). Let \(v_x\) be the first\(^1\) computation vertex in \(q_r\) and

\(^1\)the first computation vertex in a major path is the vertex adjacent to the source, i.e., the edge from source vertex is directed into this vertex.
\(v_y\) be the first computation vertex in \(q_s\).

Let \(PA(v_x) = s_1\), \(TA(v_x) = t_1\), \(PA(v_y) = s_2\) and \(TA(v_y) = t_2\). By construction of \(S_G\) there must exist tuples \(\langle v_u, t_1 \rangle\), \(\langle v_u, t_1 - k_1 d_{1j} \rangle\), \(\langle v_w, t_2 \rangle\) and \(\langle v_w, t_2 - k_2 d_{1j} \rangle\) such that

\[
\begin{align*}
t &= t_1 - k_1 d_{1j} = t_2 - k_2 d_{1j} \\
IOA(v_u, t_1) &= s_1 \quad \text{and} \quad IOA(v_w, t_2) = s_2 \\
IOA(v_u, t_1) &= IOA(v_w, t_2) = s = s_1 - k_1 n_{1j} = s_2 - k_2 n_{1j}
\end{align*}
\]

and hence,

\[
\begin{align*}
s_1 &= s + k_1 n_{1j} \\
s_2 &= s + k_2 n_{1j} \\
t_1 &= t + k_1 d_{1j} \\
t_2 &= t + k_2 d_{1j}
\end{align*}
\]

and hence,

\[
\begin{align*}
s_2 &= s_1 + (k_2 - k_1) n_{1j} \\
t_2 &= t_1 + (k_2 - k_1) d_{1j}
\end{align*}
\]

and so \(v_x^{T_k(a, b)} v_y\) where \(a = n_{1j}\), \(b = d_{1j}\) and \(k = k_2 - k_1\). But \(v_x\) and \(v_y\) are in distinct paths \(q_r\) and \(q_s\) respectively and we have assumed that they both are in the same major path labelled \(l_j\). We can obtain a similar contradiction in the other two cases also.

Lemma 4.0-4:

In any syntactically correct mapping of \(G\) and for any \(E_{1i}\) and \(E_{1j}\) in \(H\) if \(n_{1i} = n_{1j}\) then \(d_{1i} \neq d_{1j}\).

Proof: Since \(E_{1i}\) and \(E_{1j}\) are in \(H\) they cannot be identical giving rise to the two cases similar to the two cases considered in the proof of lemma 4.0-1. For reasons similar to the one given in proof of lemma 4.0-1 case (1) cannot arise in any syntactically correct mapping.

Consider case (2a). Let the distance from \(v_x\) to \(v_y\) in figure...
4. If \( k \) be \( k \) Now \( PA(v_y) = PA(v_x) + k\cdot n_{1j} \) and \( TA(v_y) = TA(v_x) + k\cdot d_{1j} \). Suppose \( d_{1j} = d_{1j} \) and hence \( PA(v_y) = PA(v_x) + k\cdot n_{1j} \) and \( TA(v_y) = TA(v_x) + k\cdot d_{1j} \) and hence \( v_x \xrightarrow{T^k_{a,b}} v_y \) where \( a = n_{1j} \) and \( b = d_{1j} \) and so by lemma 4.0-3 there must be a single major path labelled \( 1j \) passing through \( v_x \) and \( v_y \). But in figure 4.0-1 \( v_x \) and \( v_y \) are in \( q_s \) and \( q_t \) respectively and \( q_s \) and \( q_t \) are distinct major paths labelled \( 1j \).

Consider case (2b). We can similarly show that \( v_y \xrightarrow{T^k_{a,b}} v_z \) where \( a = n_{1j} \) and \( b = d_{1j} \). But \( v_y \) and \( v_z \) are in distinct major paths \( q_r \) and \( q_t \) labelled \( 1j \).

**Definition 4.0-3:** A maximally-connected subgraph of a uniform graph \( G \) is a 5-tuple \( SG = <V_{SG}, S_{SG}, E_{SG}, L_{SG}> \) where:

1. \( V_{SG}, S_{SG}, E_{SG}, L_{SG} \) are subsets of \( V_G, S_G, E_G, L_G \) respectively.

2. and \( SG \) satisfies the following:

   a. the edges in \( E_{SG} \) are labelled with labels from \( L_{SG} \)

   b. \( SG \) is connected

   c. there exist no pair of vertices \( v_x \) and \( v_y \) such that \( v_x \)

   is not in \( SG \) and \( v_y \) is in \( SG \) and there is a undirected

\[ \text{there is an undirected path between every pair of computation vertices in } SG \text{ such that the edges in the path are all from } E_{SG}. \]
path from $v_x$ to $v_y$ through edges whose labels are in $L_{SG}$

Example 4.0-1:

![Graphs G, SG1, SG2](image)

Figure 4.0-3

$G$ is a uniform graph. Computation vertices are denoted by ‘o’, source and sink vertices are denoted by ‘@’. The horizontal, vertical and oblique (curved edges in $G$) are labelled 11, 12 and 13 respectively. The source and sink vertices attached to horizontal, vertical and oblique edges are labelled 11, 12 and 13 respectively. $SG_1$ and $SG_2$ are the two maximally-connected subgraphs with $L_{SG_1}=L_{SG_2}={11, 12}$.

Let $l_i$ and $l_j$ be any two labels in $L_G$ such that $E_{l_i}$ and $E_{l_j}$ are in $H$. Let $SG$ be a maximally-connected subgraph with $L_{SG}={l_i, l_j}$. Let $SG_{l_i}=$ {major paths in $SG$ labelled $l_i$} and $SG_{l_j}=$ {major paths in $SG$ labelled $l_j$}. Now $SG_{l_i}$ and $SG_{l_j}$ are subsets of $E_{l_i}$ and $E_{l_j}$ respectively.

Let $S_{l_i}=$ {$r_{l_i}$ | $l_i$ is a label and $r_{l_i}$ is binary relation on major paths in

\[1\] recall definition 2.1-6, page 7
$SG_{11}$. Since $L_{SG} = \{1i, 1j\}$ and hence $S_{1i} = \{r_{1i}\}$. Similarly $S_{1j} = \{r_{1j}\}$. For any mapping of $G$ to be syntactically correct $r_{1i}$ and $r_{1j}$ must impose a structure on the major paths in $SG_{11}$ and $SG_{1j}$ which is stated formally in the following lemma.

Lemma 4.0-5:

For any mapping of $G$ to be syntactically correct $r_{1j}$ in $S_{1j}$ and $r_{1i}$ in $S_{1j}$ must impose a linear chain on the major paths in $SG_{1j}$ and $SG_{1j}$ respectively.

Proof: We will first show that $r_{1j}$ in $S_{1j}$ must impose a linear chain on major paths in $SG_{1j}$. In order to show this we must first show that $S_{1j}$ imposes a consistent order on major paths in $SG_{1j}$.

Let $G_s$ and $G_x$ be the directed graph induced by all the relations in $S_{1j}$ and by the relation $r_{1j}$ respectively on the major paths in $SG_{1j}$. $S_{1j} = \{r_{1j}\}$ and hence $G_s = G_x$. Consequently inorder to show that $S_{1j}$ imposes a consistent order on the major paths in $SG_{1j}$ we must show that $r_{1j}$ is total on $SG_{1j}$ and $G_x$ is acyclic.

Suppose $r_{1j}$ is not total. This means there exist atleast two major paths $q_s$ and $q_t$ in $SG_{1j}$ such that neither $q_s \uparrow r_{1j} \downarrow q_t$ nor $q_t \uparrow r_{1j} q_s$. Now $SG$ is connected and hence $G_x$ is connected and so there must be a major path $q_k$ in $SG_{1i}$ such that one of the following two cases hold:
1. \( q_s r_{ij}^+ q_k, q_t r_{ij}^+ q_k \) and there does not exist any major path \( q_w \) in \( SG_{11} \) such that \( q_s r_{ij}^+ q_w, q_t r_{ij}^+ q_w \) and \( q_w r_{ij}^+ q_k \)

2. \( q_k r_{ij}^+ q_s, q_k r_{ij}^+ q_t \) and there does not exist any major path \( q_w \) in \( SG_{11} \) such that \( q_k r_{ij}^+ q_w, q_w r_{ij}^+ q_s \) and \( q_w r_{ij}^+ q_t \)

Consider the first case. Let \( q_m \) and \( q_n \) be two major paths in \( SG_{11} \) such that \( q_s r_{ij}^+ q_m, q_t r_{ij}^+ q_n, q_m r_{ij}^+ q_k \) and \( q_n r_{ij}^+ q_k \).

Since \( q_w \) in (1) and (2) is non-existent and hence \( q_m \) and \( q_n \) are two distinct major paths. \( q_m r_{ij}^+ q_k \) and so there must exist a pair of computation vertices \( v_x \) and \( v_w \) in \( q_m \) and \( q_k \) respectively such that there is an edge \( e_{a} \) labelled \( lj \) directed from \( v_x \) to \( v_w \). Similarly since \( q_n r_{ij}^+ q_k \) and hence there must exist a pair of computation vertices \( v_y \) and \( v_u \) in \( q_n \) and \( q_k \) respectively such that there is an edge \( e_{b} \) labelled \( lj \) directed from \( v_y \) to \( v_u \).

This is illustrated in figure 4.0-4 below:

![Figure 4.0-4](image)

In Figure 4.0-4 each of the shaded boxes denote a major path labelled \( li \). Let the distance from \( v_w \) to \( v_u \) in \( q_k \) be \( h \) and hence in any syntactically correct mapping,
\[ \text{PA}(v_u) = \text{PA}(v_w) + h \times n_{1i} \]

and \[ \text{TA}(v_u) = \text{TA}(v_w) + h \times d_{1i} \]

Now \( v_x \xrightarrow{r_{1j}} v_w \) and hence,
\[ \text{PA}(v_w) = \text{PA}(v_x) + n_{1j} \]

and \[ \text{TA}(v_w) = \text{TA}(v_x) + d_{1j} \]

Also \( v_y \xrightarrow{r_{1j}} v_u \) and hence,
\[ \text{PA}(v_u) = \text{PA}(v_y) + n_{1j} \]
\[ \text{TA}(v_u) = \text{TA}(v_y) + d_{1j} \]

From the above equations we obtain,
\[ \text{PA}(v_y) = \text{PA}(v_x) + h \times n_{1i} \]
\[ \text{TA}(v_y) = \text{TA}(v_x) + h \times d_{1i} \]

Hence \( v_x \xrightarrow{<a, b>} v_y \) where \( a = n_{1i} \) and \( b = d_{1i} \). If \( n_{1j} = \emptyset \) then \( \text{PA}(v_x) = \text{PA}(v_y) \) and by definition of a syntactically correct mapping \( v_x \) and \( v_y \) must both be in the same major path labelled \( l_i \). If \( n_{1j} \neq \emptyset \) then by lemma 4.0-3 \( v_x \) and \( v_y \) must both be in the same major path labelled \( l_i \). But \( v_x \) and \( v_y \) are in distinct major paths \( q_m \) and \( q_n \). We can arrive at a similar contradiction in the second case also.

We must next show \( G_x \) is acyclic. Suppose there is a cycle in \( G_x \). Let \( q_1, q_2, \ldots, q_m \) be the set of the \( m \) major paths in \( S G_{1i} \) that form a cycle in \( G_x \) as shown in figure 4.0-5.
This cycle implies that there are two computation vertices \( v_x \) and \( v_y \) in \( q_1 \) such that there is a undirected path from \( v_x \) to \( v_y \) that passes through computation vertices in each of \( q_2, q_3, \ldots, q_m \) and through edges labelled \( l_i \) or \( l_j \) as shown in figure 4.0-6.

Let \( k_1 \) and \( k_2 \) be the number of edges labelled \( l_i \) in this path such that the order on the vertices imposed by these edges are consistent and not consistent respectively with the order on the vertices imposed by the undirected path. Let \( k_1-k_2=h \). The number of edges in this path that are labelled \( l_j \) is \( m-1 \) and clearly \((m-1)>1\). By proposition 4.0-1,

\[
PA(v_y) = PA(v_x) + h*n_{11} + (m-1)*n_{1j}
\]
and \( TA(v_y) = TA(v_x) + h * d_{ij} + (m-1) * d_{ij} \)

Let the distance from \( v_x \) to \( v_y \) in the major path \( q_1 \) be \( k \) and hence,

\[ PA(v_y) = PA(v_x) + k * n_{ij} \]

and \( TA(v_y) = TA(v_x) + k * d_{ij} \)

and so,

\[ k * n_{ij} = h * n_{ij} + (m-1) * n_{ij} \] \[ \text{(a)} \]

\[ k * d_{ij} = h * d_{ij} + (m-1) * d_{ij} \] \[ \text{(b)} \]

Now \( E_{ij} \) and \( E_{lj} \) are in \( H \) and hence by proposition 4.0-5 each of \( n_{ij} \) and \( n_{lj} \) must be one of \( \{1, -1, \emptyset\} \). By lemma 4.0-1 \( n_{ij} = n_{lj} \) \( \text{iff} \) and so the possible values that the tuple \( \langle n_{ij}, n_{lj} \rangle \) can assume are: \( \langle 1, \emptyset \rangle, \langle 1, 1 \rangle, \langle 1, -1 \rangle, \langle -1, \emptyset \rangle, \langle -1, 1 \rangle, \langle -1, -1 \rangle, \langle \emptyset, 1 \rangle \) and \( \langle \emptyset, -1 \rangle \).

1. Consider the set of values \( \langle 1, 1 \rangle \) and \( \langle -1, -1 \rangle \). From (a) and (b) \( d_{ij} = d_{lj} \). But by lemma 4.0-4 \( d_{ij} = d_{lj} \).

2. Consider the set of values \( \langle \emptyset, 1 \rangle \) and \( \langle \emptyset, -1 \rangle \). From (a) \( n_{lj} = \emptyset \).

3. Consider the set of values \( \langle 1, \emptyset \rangle \) and \( \langle 1, \emptyset \rangle \). From (a) and (b) \( d_{lj} = \emptyset \).

4. Consider the set of values \( \langle -1, 1 \rangle \) and \( \langle 1, -1 \rangle \). From (a) and (b) \( d_{ij} = d_{lj} \).
and hence $G_X$ must be acyclic.

Lastly we must show that the consistency constant of $r_{1j}$ in $S_{11}$ is 1. Suppose otherwise. This means there exist major paths $q_s$ and $q_t$ in $S_{11}$ such that $q_s r_{1j}^m q_t$ and $q_s r_{1j} q_t$ in $G_X$ and $m > 1$ as shown in figure 4.8-7.

![Figure 4.8-7](image)

Now $q_s r_{1j} q_t$ and hence there must exist computation vertices $v_x$ and $v_y$ in $q_s$ and $q_t$ respectively such that there is an edge $e_b$ labelled $1j$ directed from $v_x$ to $v_y$. Also $q_s r_{1j}^m q_t$ and so there must exist major paths $q_{s1}$, $q_{s2}$, ..., $q_{s(m-1)}$ in $S_{11}$ such that $q_s r_{1j} q_{s1}$, $q_{s1} r_{1j} q_{s2}$, ..., $q_{s(m-1)} r_{1j} q_t$ and hence there must exist an undirected path from $v_x$ to $v_y$ through computation vertices in each of $q_{s1}$, $q_{s2}$, ..., $q_{s(m-1)}$ and through edges labelled $li$ or $lj$ as shown in figure 4.8-8.

![Figure 4.8-8](image)

Let $k_1$ and $k_2$ be the number of edges labelled $li$ in the
undirected path from $v_x$ to $v_y$ such that these edges impose an order on the vertices in the path that are consistent and not consistent respectively with the order imposed by the path on these vertices. The number of edges labelled $l_j$ in this path is $m$ and hence,

$$PA(v_y) = PA(v_x) + m \cdot n_{l_j} + h \cdot n_{l_1}$$
and $$TA(v_y) = TA(v_x) + m \cdot d_{l_j} + h \cdot d_{l_1}$$
also $q_s r_{l_j} q_t$ and hence

$$PA(v_y) = PA(v_x) + n_{l_j}$$
$$TA(v_y) = TA(v_x) + d_{l_j}$$
and so, $h \cdot n_{l_1} = (m-1) \cdot n_{l_j}$
and $h \cdot d_{l_1} = (m-1) \cdot d_{l_j}$

By reasons similar to that used in showing $G_x$ is acyclic we conclude that the consistency constant of the relation $r_{l_j}$ in $S_{l_1}$ must be 1. Hence $r_{l_j}$ in $S_{l_1}$ must impose a linear chain on the major paths in $SG_{l_1}$. Similarly we can prove that $r_{l_1}$ in $S_{l_j}$ must impose a linear chain on the major paths in $SG_{l_j}$.

5. Syntactic Characterization

In this section we provide a characterization theorem for a class of program graphs such that there exists a syntactically correct mapping for every member in this class. As a prelude to the main result (theorem 5.2-1) we establish some preliminary results.
5.1. Mesh-graphs

Let $\text{MG}$ be a uniform graph such that $L_{\text{MG}} = \{ l_l, l_j \}$ and $r_{ll}$ and $r_{lj}$ are both non-empty.

Definition 5.1-1: $\text{MG}$ is a mesh graph iff there is a one-one function $F: V_{\text{MG}} \rightarrow I \times I$ where:

1. $I$ is a set of integers

2. $V_{\text{MG}}$ is the set of computation vertices in $\text{MG}$

3. $F_{ll}$ and $F_{lj}$ are projection functions of $F$, i.e., if $F(v_x) = \langle m, n \rangle$ then $F_{ll}(v_x) = m$ and $F_{lj}(v_y) = n$

4. for any $v_x$ and $v_y$ in $V_{\text{MG}}$
   a. if there exists a major path labelled $ll$ passing through $v_x$ and $v_y$ such that the distance from $v_x$ to $v_y$ in this major path is $k$ then $F_{ll}(v_y) = F_{ll}(v_x) + k$ and $F_{lj}(v_y) = F_{lj}(v_x)$ and conversely so¹

   b. if there exists a major path labelled $lj$ passing through $v_x$ and $v_y$ such that the distance from $v_x$ to $v_y$ in this major path is $k$ then $F_{lj}(v_y) = F_{lj}(v_x) + k$ and $F_{ll}(v_y) = F_{ll}(v_x)$ and conversely so.

¹ i.e., if $F_{ll}(v_y) = F_{ll}(v_x) + k$ and $F_{lj}(v_y) = F_{lj}(v_x)$ then there must exist a major path labelled $ll$ passing through $v_x$ and $v_y$ and the distance from $v_x$ to $v_y$ in this path must be $k$. 
We will denote \( F(v_x) \) by the tuple \( \langle x_{11}, x_{1j} \rangle \) where \( F_{11}(v_x) = x_{11} \) and \( F_{1j}(v_x) = x_{1j} \). We can think of \( x_{11} \) and \( x_{1j} \) as the horizontal and vertical co-ordinates of the computation vertex \( v_x \).

**Example 5.1-1:**

![Diagram of Mesh Graph MG](image)

**Figure 5.1-1: Mesh Graph MG**

In figure 5.1-1 'o' denote computation vertices and 'θ' denote source and sink vertices.

\[ F(v_1) = \langle \emptyset, \emptyset \rangle, \quad F(v_2) = \langle 1, \emptyset \rangle, \quad F(v_3) = \langle \emptyset, 1 \rangle, \quad F(v_4) = \langle 1, 1 \rangle, \quad F(v_5) = \langle 2, 1 \rangle \text{ and } F(v_6) = \langle \emptyset, 2 \rangle. \]

We can characterize a mesh graph in terms of the relations \( r_{11} \) and \( r_{1j} \) on the major paths in \( E_{1j} \) and \( E_{11} \) respectively. Consider a uniform graph \( MG \) with \( L_{MG} = \{l_1, l_j\} \) and \( r_{11} \) and \( r_{1j} \) are non-empty.

**Proposition 5.1-1:**

\( MG \) is a mesh graph iff each of the following two conditions hold:

1. \( r_{1j} \) imposes a linear chain on the major paths labelled \( l_1 \) in \( MG \).
2. \( r_{11} \) imposes a linear chain on the major paths labelled \( lj \) in MG.

If MG satisfies proposition 5.1-1 then the major paths in \( E_{ll} \) are totally ordered and similarly the major paths in \( E_{lj} \) are also totally ordered. So if \( v_x \) is a computation vertex in MG then \( F(v_x) = <m,n> \) where \( m \) is the index of the major path in \( E_{lj} \) which passes through \( v_x \) and \( n \) is the index of the major path in \( E_{ll} \) that passes through \( v_x \).

Using proposition 5.1-1 we can characterize the property of any maximally-connected subgraph of a uniform graph \( G \) inorder that any mapping of \( G \) is syntactically correct.

**Theorem 5.1-1**: For any pair of labels \( li \) and \( lj \) in \( L_G \) of a uniform graph \( G \) if \( SG \) is any maximally-connected subgraph of \( G \) with \( L_{SG} = \{li,lj\} \) then \( SG \) must be a mesh graph in order for any mapping of \( G \) to be syntactically correct.

**Proof**: Immediate from lemma 4.0-5 and proposition 5.1-1.

In any syntactically correct mapping of a mesh graph we can easily relate the processors computing any pair of computation vertices and also the times at which they are computed.

**Proposition 5.1-2**: In any syntactically correct mapping of the mesh graph MG and for any pair of computation vertices \( v_x \) and \( v_y \),

\[
PA(v_y) = PA(v_x) + (y_{1i} - x_{1i}) \times n_{1i} + (y_{1j} - x_{1j}) \times n_{1j}
\]