Space and Time Hierarchies for Control Structures and Data Structures

R. J. Lipton, S. C. Eisenstat, and R. A. DeMillo[†]

Research Report #61

December 1975

Some of these results were presented at the 7th ACM Symposium on the Theory of Computing, Albuquerque, New Mexico, May 1975. The research was supported in part by the US Army Research Office under Grants DAHC04-74-G-0179 and DAHC04-75-G-0037 and by the Office of Naval Research under Grant N00014-67-0097-0016. Some of this work was carried out while DeMillo and Lipton were visitors at the Communications/ADP Laboratory of the US Army Electronics Command, Fort Monmouth, New Jersey.

† University of Wisconsin, Milwaukee, Wisconsin 53201.

Abstract

Control structures and data structures are modelled by directed graphs. In the control case, nodes represent executable statements and arcs represent possible flow of control; in the data case, nodes represent memory locations and arcs represent logical adjacencies in the data structure. Classes of graphs are compared by a relation $\leq_{S,T}$ where

$$G \leq_{S,T} H$$

if G can be embedded in H with at most a T-fold increase in distance between embedded nodes by making at most S "copies" of any node in G. For both control structures and data structures, S and T are interpreted as space and time constants respectively. Results are presented that establish hierarchies with respect to $\leq_{S,T}$ for (1) data structures, (2) sequential program schemata normal forms, and (3) sequential control structures.

Key Words and Phrases: ancestor tree, bounded simulation, complexity, control structure, data structure, directed graph, do forever program, embedding, goto program, label exit program, normal form programs, structured programming, while programs.

CR Categories: 4.22, 4.34, 5.24, 5.25, 5.32.

1. Introduction

The running time or computational complexity of a sequential process is usually estimated by summing weights attached to the basic operations from which the process is derived. In practice, however, the complexity of a program is often limited by how efficiently it can access its data structures and control program flow. Furthermore, it has been extensively argued [4] that certain limitations on the process sequencing mechanisms available to the programmer result in more "efficient" representations for the underlying processes. In this paper we will examine these issues in an attempt to assess the "power" of various data and control structures.

A key observation about sequential processes is that they usually do not reference their data randomly. For instance, algorithms that organize their data structures as arrays often access the array elements in a "local" manner (e.g. the conventional matrix multiplication algorithm accesses its arrays by rows and columns). Thus, in a paging environment, how one stores an array is especially important (cf. Moler [13], Rosenberg [16]), and it is natural to investigate how arrays can be stored so that elements "near" one another in the array are stored near one another in memory. Data structures are compared by the relation $\leq_{1,T}$: For data structures G and G*, G $\leq_{1,T}$ G* if G can be embedded in G* so that there is at most a T-fold increase in distance between embedded objects.

It is somewhat unexpected that an analogous study for control structures uses the same basic insights. It is well known that process sequencing disciplines found in programming practice (e.g. goto, while) can *simulate* each other and are thus equivalent in the sense of yielding *functionally* equivalent programs, but are inequivalent relative to the stronger requirement of structural isomorphism [1,2,3,10,11]. We argue that the fundamental issue is neither the construction of functionally equivalent programs nor the inability to preserve structure exactly, but rather the "naturalness" of the simulation. Control structures are compared by the relation $\leq_{S,T}$: For algorithms G and G* with distinct process sequencing mechanisms, $G \leq_{S,T} G^*$ if G^* simulates G by making at most S copies of each operation in G and increasing the cost of sequential access of embedded operations by a factor of at most T.

Thus, comparing the power of data structures and control structures involves analyzing the one-one and many-one aspects of reduction (or simulation) techniques whose efficiency is bounded by S and T. In a natural way, the relation $\leq_{S,T}$ represents an intertwining of space and time complexities.

The plan of presentation is as follows. In Section 2, the basic combinatorial definitions used throughout the sequel are presented, and the combinatorial models used for representing data structures and control structures are introduced. In Section 3, the relation $\leq_{S,T}$ is defined by means of graph embeddings. This relation is viewed as an embedding in the data structure case and as a simulation relation in the control structure case. Section 4 contains the main result for data structure embeddings: that for certain families of structures $\{G_i\}_{i\geq 0}$ and $\{G_i^*\}_{i\geq 0}$, if $G_i \leq_{1,T} G_i^*$, then

$$T \geq c \cdot \log n_i \quad \dagger$$

for some positive constant c whose choice is independent of n_i, the number of nodes of G_i.

The main theorem in Section 5 generalizes the result in Section 4 by allowing $S \geq 1$. In this case, if $G_i \leq_{S,T} G_i^*$ for certain natural choices of $\{G_i\}_{i\geq 0}$ and $\{G_i^*\}_{i\geq 0}$, then

$$T + \log S \geq c \cdot \log n_i$$

where n_i is the number of nodes of G_i and c is a positive constant independent of n_i. A direct result of this theorem is that certain schema constructions, such as Engeler

† When we establish results of this form, we are asserting that there is a minimal rate of growth for T as a function of n_i. In the sequel we will consistently abuse our notation by writing $f(S,T) \geq g(n)$ instead of the less convenient $f(S(n),T(n)) \geq g(n)$. It will usually be clear from context when S,T are to be considered constants and when S,T are parameterized.

normal form [6], cannot be achieved "uniformly" with respect to the $\leq_{S,T}$ relation. More exactly, for any constants S and T there is a goto program G such that for no program H in Engeler normal form is $G \leq_{S,T} H$. Thus, the construction of Engeler normal forms -- while always possible -- does not preserve time and space in a bounded way. This result also demonstrates how our results will be asymptotic in their nature: For any goto program G there are S and T such that $G \leq_{S,T} H$ where H is the Engeler normal form; however, the values of S and T must grow with the size of the program G.

In Section 6, the relation $\leq_{S,T}$ is placed in the context of relations used in previous studies of control structure simulation. The main simulation results for control structures are then developed, giving rise to the hierarchy of control structures shown in Figure 1. An important result is that goto programs are strictly more powerful than label exit programs. Since the class of label exit programs includes many of the standard constructs that are allowed in "structured" programs, this result can be viewed as a precise sense in which there is a *time-space speedup* between goto programs and "structured" programs: There are goto programs whose only "structured" counterparts explode in either time or space. This result seems to make precise the comments of Knuth [9] on the efficiency of goto and "structured" programs.

While the results in this paper are motivated by our interest in the power of data and control structures, they may have interest purely as combinatorial results.

2. The Combinatorial Representations

A *directed graph* G is an ordered pair (V,E) of *nodes* and *arcs*. If there is an arc from x to y and an arc from y to x, then we will say there is an *edge* between x and y. Moreover, the arcs shown in Figure 2a will be represented as in Figure 2b. A *path* from x to y is defined by any sequence of arcs from $x = x_0$ to x_1 to x_2 to ... to $x_n = y$. We define the *metric* $d_G(x,y)$ on G as the number of arcs in a minimal length path from x to y.

A *binary tree* is a directed graph that either is a single node or consists of a root x and an edge between x and the root of each of two binary trees called the left and right *subtrees* of the root (cf. Knuth [8]). Note that nodes in a binary tree are connected by *edges* so that the metric is symmetric. If G is a binary tree, then a node x of G is a *leaf* of G if x has no sons.

We will represent both control structures and data structures by directed graphs. In ths control case, the nodes of a graph G represent executable statements and the arcs represent possible flow of control; in the data case, the nodes of the graph represent memory locations and the arcs, logical adjacencies in the data structure. Thus, in either case, what is to be modelled is the "difficulty" of accessing nodes: The complexity of a control structure† is given by the cost of accessing and sequencing non-control instructions, while the complexity of a data structure is determined by the cost of accessing successive data elements. Each class of control structure or data structure will then be studied in terms of restrictions on what graphs are allowed in that representing class.

† Some care must be exercised in viewing control structures that are represented in this way; our representations do not always correspond to (temporal) flow of control and are not to be looked at as flowcharts. Rather, what is being modelled is the *potential* control connectivity of an underlying algorithm or process.

2.1. Data Structures

The two classes of data structure we will be dealing with are *arrays* and *ancestor trees.*

Arrays: G_n will denote the data structure corresponding to an $n{\times}n$ array. If the nodes of G_n are indexed by (i,j) where $1 \le i \le n$ and $1 \le j \le n$, then there is an *edge* between (i,j) and $(i,j+1)$ and between (i,j) and $(i+1,j)$. Thus G_n is "rook connected." For instance, G_3 is illustrated in Figure 3.

Ancestor Trees: Ancestor trees are binary trees with an additional feature: A node x of an ancestor tree may be connected by an *arc* to any of its ancestors. For example, the graph shown in Figure 4 is an ancestor tree because y is both an ancestor *and* a successor of x. Notice, however, that unlike metrics on binary trees, the graph metric d_G is not necessarily symmetric on ancestor trees. Ancestor trees include linear lists, circular lists, and threaded lists [14] as special cases.

2.2. Control Structures

We will consider the following five classes of control structures:

computed **goto**
goto with d-way branching
label **exit**
do forever
while.

In addition, all of the available classes will have access to a sequential flow of control and an alternative (e.g. **if-then-else**) flow of control. Since the constructions described below do not involve schema manipulation, the details of these features need not be made explicit. Indeed, there are a number of ways to represent these features in our model, and our later results are invariant under the differing representations. We will now present the class of graphs that represent programs formed from each of the five control structures.

Computed goto Programs: goto_ω programs are programs that allow arbitrary branching between statements. For instance, we allow for representations of the construct

$$\text{goto } i \ (L_1, \ldots, L_n),$$

which branches to the *ith* label depending on the value of i. Thus, this class of programs is represented by the entire class of directed graphs. The construct above is represented by the graph of Figure 5: a node with n arcs leading to nodes labelled by $L_1, \ldots, L_n$.

goto Programs with d-way Branching: goto_d programs are programs in which the amount of branching that is possible in one step is bounded by the integer d. For example, the FORTRAN construct

$$\text{IF (E) } L_1, L_2, L_3$$

falls in the class goto_3. Programs with d-way branching are represented by the class of directed graphs with maximum out-degree d.†

Label exit, do forever, and while Programs: Label exit, do forever, and while programs are defined as certain classes of ancestor trees. In order to define these classes, we need the following relations, which are defined for any ancestor tree:

$x \xrightarrow[p]{} y$ if y is the left son of x.

$x \xrightarrow[s]{} y$ if y is the right son of x.

† The out-degree of a node x is $|\{y\colon d_G(x,y) = 1\}|$.

$x \xrightarrow[a]{} y$ if y has an ancestor pointer from x.

We view $x \xrightarrow[p]{} y$ as meaning that statement x can "push" into a substructure with first statement y; $x \xrightarrow[s]{} y$ as meaning that statement x is "sequentially" followed by y; and $x \xrightarrow[a]{} y$ as meaning that statement x can "exit" some structure and return to statement y.

A program is a while program provided it is an ancestor tree that satisfies: $y \xrightarrow[a]{} x$ implies $\exists\, y_1,\ldots,y_k$ such that $x \xrightarrow[p]{} y_1 \xrightarrow[s]{} \cdots \xrightarrow[s]{} y_k = y$, where y is a leaf and no y_i for $i < k$ has an ancestor pointer. (See Figure 6a.) The last restriction reflects the fact that only the last statement in a while loop is allowed to exit the loop.

A program is a do forever program provided it is an ancestor tree that satisfies: $y \xrightarrow[a]{} x$ implies $\exists\, y_1,\ldots,y_k = y$ such that $x \xrightarrow[p]{} y_1 \xrightarrow[s]{} \cdots \xrightarrow[s]{} y_k$, where each y_i can have ancestor pointers only to x. (See Figure 6b.) The key distinction between while programs and do forever programs is that, in a do forever program, all statements in a loop can potentially exit immediately out of the looping structure. Clearly, do forever programs correspond to the Ω_n $(n \geq 1)$ structures of Böhm and Jacopini [2].

A label exit program is any program that is also an ancestor tree. Essentially label exit programs allow any jumping out of substructures as long as the return is always to an ancestor. The class of label exit programs is, therefore, quite extensive and includes many types of "structured" programs (cf. Peterson et al. [15]). For example, all label exit programs are reducible in the sense of [7]; moreover, they correspond to programs in Engeler normal form [6].

Example: The following program contains label exit, do forever, and while control structures; its representation using the conventions outlined above is shown in Figure 7.

```
L:  $S_1$;
    while $B_1$ do
       begin $S_2$;
          do forever
             begin $S_3$; exit L;
                do forever begin $S_4$; exit; $S_5$ end;
                $S_6$
             end;
          $S_7$
       end;
    $S_8$;   □
```

3. S,T Bounded Reductions

The following definition is fundamental to what follows. Let $G = (V,E)$ and $G^* = (V^*,E^*)$ be directed graphs with associated metrics d_G and d_{G^*}. We say that G^* can *simulate* G (or G can be *reduced* to G^*) with space constant S and time constant T, written

$$G \leq_{S,T} G^*,$$

if there is a mapping (called an *embedding*) $\Phi: V^* \rightarrow V \cup \{\Lambda\}$ of the nodes of G^* to the nodes of G and a special node Λ, so that:

1) $\forall v^* \in V^*$ with $\Phi(v^*) \neq \Lambda$

   $\forall w \in V$ such that $d_G(\Phi(v^*),w) < \infty$

   $\exists w^* \in V^*$ such that $\Phi(w^*) = w$ and $d_G(v^*,w^*) \leq T \cdot d_G(\Phi(v^*),w)$;

2) $\forall v \in V,\ 0 < |\Phi^{-1}(v)| = |\{v^* \in V^*: \Phi(v^*) = v\}| \leq S.$

If Φ is an embedding and $\Phi(v^*) = \Lambda$, then we will refer to v^* as a *bookkeeping* node. If $\Phi(v^*) = v \neq \Lambda$, then v^* is said to be a *copy* of v. If $S = 1$, we will often write $\leq_T$ instead of $\leq_{1,T}$.

Condition (1) states that when G and G^* are control structures (or data structures) simulation involves at most a T-fold increase in the cost of statement sequencing (or data element accessing); i.e. the embedding induces at most a T-fold increase in path length. Condition (2) states that there are at most S copies of any $v \in V$ in G^*. Note that, although $G \leq_{S,T} G^*$ may hold between data structures G and G^* when $S > 1$, it is unlikely that such a simulation would be of value (e.g. if an array is being stored as a list structure with multiple copies of array elements, then selective updating of the array may involve multiple updating of list nodes). For control structures, however, simulations with $S > 1$ are frequently used and are quite natural; this is sometimes called *node splitting*.

__Example:__ Consider the flow diagrams shown in Figure 8. Figure 8b is the result of applying a standard "restructuring" algorithm [1] to 8a to remove the multiple exit loop x_3, x_4, x_5. Viewing both diagrams as directed graphs, the graph in 8b is a 2,2 simulation of 8a by defining Φ as follows:

$$\Phi(x_1^*) = x_1$$
$$\Phi(x_2^*) = x_2$$
$$\Phi(x_3^*) = \Phi(x_7^*) = \Phi(x_{10}^*) = \Lambda$$
$$\Phi(x_4^*) = \Phi(x_8^*) = x_3$$
$$\Phi(x_5^*) = \Phi(x_9^*) = x_4$$
$$\Phi(x_6^*) = x_5$$
$$\Phi(x_{11}^*) = x_6$$
$$\Phi(x_{12}^*) = \Phi(x_{13}^*) = x_7$$
$$\Phi(x_{14}^*) = \Phi(x_{15}^*) = x_8$$
$$\Phi(x_{16}^*) = x_9. \quad \square$$

4. Data Structure Embeddings

In this section we will present our main result for data structures, settling negatively the question whether arrays can be stored as arbitrary lists with linear bounds on proximity and determining a nontrivial lower bound on the growth rate of T as a function of n for an n×n array. This result generalizes a result of Rosenberg's [16] showing that arrays cannot be stored in linear memory with only bounded loss of proximity. But since the arguments are fundamentally different, it is interesting to compare the two proofs. Recall that Rosenberg's arguments are essentially "volumetric": The number of neighbors within distance n of a node in an array can be quadratic in n, while a node in a linear list can have at most 2n such neighbors. A volumetric argument then demonstrates that arrays cannot be stored in a system with such linear neighborhood structure with only bounded loss of proximity. In contrast, these methods do not seem to apply to our problems; e.g. a node in a binary tree can have more than 2^n neighbors within distance n.

To obtain our result we need a series of lemmas. Let G = (V,E) be a directed graph with associated metric d_G and suppose $A \subseteq V$. We define the *boundary* of A as follows:

$$\partial(A) = \{y \in A: \exists x \notin A \text{ such that } d_G(x,y) = 1\}.$$

In other words, $\partial(A)$ is the set of nodes in A reachable from some node not in A by an arc of G.

Lemma 4.1: Let $G_n = (V_n,E)$ be an n×n array and suppose that $A \subseteq V_n$ is such that $|A| \le \frac{n^2}{2}$. Then

$$|A| \le 2|\partial(A)|^2.$$

Proof: We assume $|A| > 0$, since otherwise the lemma is trivially true, and let $A_1,\ldots,A_n$ be the *columns* of A; that is, if $\{(1,i),(2,i),\ldots,(n,i)\}$ is the i*th* column of G_n, then A_i is that subset of the column that is included in A. Let k be the number of columns A_i

such that $|A_i| \leq n$, and let $\ell \leq k$ be the number of columns A_i with $0 < |A_i| < n$. Since $|A| \leq \frac{n^2}{2}$, it follows that $(n-k)n \leq \frac{n^2}{2}$ and hence

$$k \geq \frac{n}{2}. \tag{1}$$

Notice that if $0 < |A_i| < n$ then at least one node in A_i is adjacent to a node not in A and thus contributes at least one node to $\partial(A)$; therefore

$$|\partial(A)| \geq \ell. \tag{2}$$

Suppose that $|A_{i_0}| = 0$ for some i_0, $1 \leq i_0 \leq n$. We then claim

$$|\partial(A)| \geq \max_j |A_j|. \tag{3}$$

To show this, let A_j be maximal in size and assume $i_0 < j$, the case $j < i_0$ being handled symmetrically. Select any row r of G_n such that $(r,j) \in A_j$. Now, $(r,i_0) \notin A$ by assumption, but some one or more of $(r,i_0+1),\ldots,(r,j)$ is in A. Therefore, each row r of G_n for which $(r,j) \notin A_j$ contributes at least one node to $\partial(A)$, which establishes (3).

To complete the proof of the lemma we consider two cases.

I. No A_i is empty. In this case, $\ell = k$ and, by combining (1) and (2),

$$2|\partial(A)|^2 \geq 2\ell^2 = 2k^2 \geq \frac{n^2}{2} \geq |A|.$$

II. Some A_i is empty. Let $c_1,\ldots,c_m$ denote the cardinality of the nonempty columns A_j. If some $c_p = n$, then the result follows directly from (2). If not, then $m = \ell$ and $c_1 + \ldots + c_m = |A|$, so that

$$\max_j |A_j| \geq \frac{|A|}{\ell}.$$

By (2) and (3) it follows that

$$2|\partial(A)| \geq \ell + \frac{A}{\ell}.$$

The lemma is now immediate by calculation. $\square$

Lemma 4.2: Let $G_n = (V_n,E)$ and suppose $x,y \in V_n$; then $d_{G_n}(x,y) \le 2n$.

Proof: This is an elementary property of arrays. $\square$

Lemmas 4.1 and 4.2 and the fact that $|V_n| = n^2$ summarize the basic properties of arrays that will be used in the proof of our main result.

Lemma 4.3: Let $H = (V,E)$ be an ancestor tree and let $H_0 = (V_0,E_0)$ be a subtree of H. If $x \in V_0$ and $y \in V - V_0$, then

$$d_H(y,x) \ge \text{depth of } x \text{ in } H_0.$$

Proof: Since $y \notin V_0$, any path from y to x must pass through the root of T_0. $\square$

Lemma 4.4: Let $H^* = (V^*,E^*)$ be an ancestor tree; let $H_0^* = (V_0^*,E_0^*)$ be a subtree of H^*; and let $A = \Phi(V_0^*) - \{\Lambda\}$. If $G_n \le_T H^*$ and $|A| \le \frac{n^2}{2}$, then

$$T \ge \frac{1}{2}(\log |A| - 1);$$

in other words,

$$|A| \le 2^{2T+1}.$$

Proof: Assume that $|\partial(A)| > 2^T$. Since the root of H_0^* has at most 2^T descendants of depth less than T+1, there is a node $x^* \in V_0^*$ of depth $\ge T+1$ in H_0^* such that $\Phi(x^*) \in \partial(A)$. Then there is a $y \in V_n - A$ with $d_{G_n}(y,\Phi(x^*)) \le 1$. By the definition of $\le_T$, there exists a y^* such that $\Phi(y^*) = y$ and $d_{T^*}(y^*,x^*) \le T$. Since $y \notin A$ it follows that $y^* \notin V_0^*$. But by Lemma 4.3, $d_{H^*}(y^*,x^*) \ge T+1$, which is a contradiction. Therefore, $|\partial(A)| \le 2^T$ and by Lemma 4.1 $|A| \le 2|\partial(A)|^2 \le 2^{2T+1}$. $\square$

Theorem 4.5: Let $H^* = (V^*,E^*)$ be an ancestor tree. If $G_n \le_T H^*$, then $T \ge \frac{1}{3}\log n - \frac{2}{3}$.

Proof: Assume $G_n \leq_T H^*$ and for any subtree $H_i^* = (V_i^*, E_i^*)$ of H^* let $A_i = \Phi(V_i^*) - \{\Lambda\}$. Let H_1^* and H_2^* be subtrees of some node in H^*. Either $|A_1| \leq \frac{n^2}{2}$ or $|A_2| \leq \frac{n^2}{2}$, since Φ is 1-1. Using this fact, we may assume that H^* is of the form shown in Figure 9, where $|A_i| \leq \frac{n^2}{2}$ for $1 \leq i \leq k$. (We have suppressed explicit representation of ancestral links.) Without loss of generality we assume always that the "smaller" subtree is on the right. By Lemma 4.4, $|A_i| \leq 2^{2T+1}$ for all i.

Let i be the smallest integer such that $|A_i| \neq 0$, and let j be the largest such integer. Then

$$\sum_{\ell=i}^{j} |A_\ell| \leq (j-i+1)2^{2T+1}.$$

Since $|V_n| = n^2$,

$$(j-i+1)2^{2T+1} \geq n^2. \tag{1}$$

Now let $x^* \in V_j^*$ and $y^* \in V_i^*$. Then by Lemma 4.3,

$$d_{H^*}(y^*,x^*) \geq j-i.$$

On the other hand, by Lemma 4.2,

$$d_{G_n}(\Phi(y^*),\Phi(x^*)) \leq 2n;$$

hence, since $G_n \leq_T H^*$, $d_{H^*}(y^*,x^*) \leq 2nT$. Thus

$$j-i \leq 2nT. \tag{2}$$

Combining (1) and (2) we have it that

$$2nT+1 \geq \frac{n^2}{2^{2T+1}}.$$

It follows that

$$T \geq \frac{1}{3}\log n - \frac{2}{3}. \quad \square$$

5. Main Theorem

Observe that the $S = 1$ hypothesis was used at several key points in the proof of Theorem 4.5. Since this restriction is unrealistic in dealing with control structures, we will now remove it by generalizing the previous result.

Theorem 5.1: Let $H^* = (V^*,E^*)$ be an ancestor tree and assume that $G_n \leq_{S,T} H^*$, where G_n is an $n\times n$ array. Then

$$T + \log S \geq \log n - \log 8\sqrt{2}.$$

Proof: Let Φ be the embedding function and define a new function Ψ mapping subsets of V^* to subsets of V by

$$\Psi(A^*) = \Phi(A^*) - \{\Lambda\}.$$

In other words, $\Psi(A^*)$ contains those nodes of V of which copies exist in A^*.

As in Theorem 4.5 we decompose H^* as follows. Let x_1^* be the root of H^* and write H^* as in Figure 10a, where we may assume $|\Psi(L_1^*)| \leq |\Psi(R_1^*)|$ without loss of generality. Clearly, this process can be repeated, letting x_{i+1}^* denote the root of R_i^* and expanding R_i^* at each stage of the construction. Thus, H^* can be written as in Figure 10b, where $|\Psi(L_i^*)| \leq |\Psi(R_i^*)|$ for $1 \leq i \leq k$. Notice that we have ignored all ancestral links in this construction. Indeed, we shall assume that all such links exist but suppress explicit reference to them.

Let $H_i^* = (V_i^*,E_i^*)$ denote the subtree of Figure 10c. We shall say that H_i^* is *small* if $|\Psi(V_i^*)| \leq \frac{n^2}{4}$; otherwise H_i^* is *large*. (The notion of smallness is notivated by the key to the argument in Theorem 4.5.) Let

$$D_k = \bigcup_{\substack{1\leq i\leq k \\ H_i^* \text{ small}}} \Psi(V_i^*).$$

In other words, D_k is the set of nodes in V of which copies exist in some small H_i^*.

Lemma 5.2: For some p, $\frac{n^2}{4} \le |D_p| \le \frac{n^2}{2}$.

Proof: By convention $|D_0| = 0$. If $|D_{p-1}| < \frac{n^2}{4}$ and $|D_p| \ge \frac{n^2}{4}$, then $D_p = D_{p-1} \cup \Psi(V_p^*)$, where H_p^* is small, so that

$$|D_p| \le |D_{p-1}| + |\Psi(V_p^*)| < \frac{n^2}{4} + \frac{n^2}{4} = \frac{n^2}{2}.$$

Thus we need show only that $|D_p| \ge \frac{n^2}{4}$ for some p.

Since $|\Psi(L_1^*)| \le |\Psi(R_1^*)|$ and $\Psi(V^*) = \Psi(L_1^*) \cup \{\Psi(x_1^*)\} \cup \Psi(R_1^*)$, it follows that

$$n^2 = |\Psi(V^*)| \le |\Psi(L_1^*)| + 1 + |\Psi(R_1^*)| \le 2|\Psi(R_1^*)| + 1;$$

hence

$$|\Psi(R_1^*)| \ge \frac{n^2-1}{2} \ge \frac{n^2}{4}.$$

We can obviously choose p so large that $|\Psi(R_p^*)| = 0$. We claim that the associated D_p is large.

Let i be the largest integer such that $|\Psi(R_i^*)| \ge \frac{n^2}{4}$. Since $|\Psi(R_1^*)| \ge \frac{n^2}{4}$, such an i must exist. Then $|\Psi(R_j^*)| < \frac{n^2}{4}$ for $i < j \le p$ and, since $\Psi(V_j^*) = \Psi(L_j^*) \cup \{\Psi(x_j^*)\}$, we have

$$|\Psi(V_j^*)| \le 1 + |\Psi(L_j^*)| \le 1 + |\Psi(R_j^*)| < 1 + \frac{n^2}{4}$$

Thus, H_j^* is small for $i < j \le p$. But this implies

$$\Phi(R_i^*) \subseteq \bigcup_{i<j\le p} \Psi(V_j^*) \subseteq D_p.$$

By our choice of i, however, we conclude that $|D_p| \ge |\Phi(R_i^*)| \ge \frac{n^2}{4}$, establishing our claim. □

We now introduce a variant of the concept of boundary. If A is a set of nodes of G_n, then the *coboundary* of A is defined by

$$\tilde{\partial}(A) \equiv \{y \notin A: \text{there exists } x \in A \text{ such that } d_{G_n}(x,y) = 1\} = \partial(V_n - A).$$

In other words, $\tilde{\partial}(A)$ is the set of nodes not in A reachable from some node in A in one step. The proof of the following result is similar to that of Lemma 4.1 and is omitted.

Lemma 5.3: Let A be a set of nodes of G_n with $|A| \le \frac{n^2}{2}$. Then $|A| \le 2|\tilde{\partial}(A)|^2$.

Let k satisfy Lemma 5.2. By Lemma 5.3,

$$|\tilde{\partial}(D_k)| \ge \frac{1}{\sqrt{2}} |D_k|^{\frac{1}{2}} \ge \frac{n}{2\sqrt{2}}.$$

Now let

$$\ell = |\{H_i^*: H_i^* \text{ is large}\}|,$$

the number of large subtrees. Since at most S copies of any node in G_n appear in H*,

$$\frac{\ell n^2}{4} \le \sum_{\substack{1 \le i \le k \\ H_i^* \text{ large}}} |\Psi(V_i^*)| \le Sn^2.$$

Hence $\ell \le 4S$.

In order to complete the proof our plan is as follows: We have already shown that

$$|\partial(D_k)| \ge \frac{n}{2\sqrt{2}};$$

we will show next that this implies that there are too many paths into the large trees H_i^* from the small trees for S and T to be bounded.

Let

$$Q_T = \{v^* \in V_i^*: H_i^* \text{ is large and there exist } H_j^* \text{ small and } x^* \in V_j^* \text{ such that } d_{H^*}(x^*,v^*) \le T\}.$$

In other words, Q_T is the set of nodes in large subtrees H_i^* that can be reached from some node in some small subtree H_j^* in at most T steps. We define a 1-1 mapping g from $\tilde{\partial}(D_k)$ into Q_T as follows. Select some $y \in \tilde{\partial}(D_k)$. Then $y \notin D_k$ and, for some $x \in D_k$, $d_{G_n}(x,y) \le 1$. Let x* be a copy of x in some small H_j^*. Such a copy exists by the definition of D_k. Since $G_n \le_{S,T} H^*$, there is a copy y* of y such that $d_{H^*}(x^*,y^*) \le T$. Now y* is not in any small H_j^* since $y \notin D_k$. Thus we can define $g(y) = y^*$ and g is indeed a mapping from

$\tilde{\partial}(D_k)$ to Q_T. In order to see that g is 1-1, we note that for any $y \in \tilde{\partial}(D_k)$, $\Phi g(y) = \Phi(y^*) = y$; hence, g is 1-1, so that $|Q_T| \geq |\tilde{\partial}(D_k)|$.

Thus we have, on the one hand, that

$$|Q_T| \geq |\tilde{\partial}(D_k)| \geq \frac{n}{2\sqrt{2}}$$

and, on the other hand, that

$$\begin{aligned}|Q_T| &\leq |\{H_i^*: H_i^* \text{ is large}\}| \cdot |\{v^*: v^* \in \text{large } H_i^* \text{ within depth T of the root of } H_i^*\}| \\ &\leq \ell \cdot 2^T \\ &\leq 4S \cdot 2^T.\end{aligned}$$

Combining the upper and lower bounds on $|Q_T|$, we deduce that

$$T + \log S \geq \log n - \log 8\sqrt{2}. \quad \square$$

As an application of Theorem 5.1, we present the following result. Informally, a flowchart is said to be in Engeler normal form if it is represented by a tree augmented by pointers from nodes to ancestors, or nodes at an earlier level but along the same branch. More precisely, a goto program G has an S,T Engeler normal form if $G \leq_{S,T} H$ for some ancestor tree H.

Corollary 5.4:

1) If G_n has an S,T Engeler normal form and T is fixed, then $S \geq c \cdot n$.

2) If G_n has an S,T Engeler normal form and S is fixed, then $T \geq c \cdot \log n$.

Thus, in the worst case, either time or space must be unbounded in the construction of Engeler normal forms.

6. Control Structures

In this section we establish our main results for control structures, using the relation $\le_{S,T}$. (See Figure 1.) For classes X and Y of control structures (i.e. classes of graph representations of programs constructed using only the indicated restricted class of control structures), we say that X is *more powerful* than Y (X > Y) where there exist S',T' such that

(1) for all $H \in Y$ there exists $G \in X$ such that $H \le_{S',T'} G$

but for no constants S,T is it true that

(2) for all $G \in X$ there exists $H \in Y$ such that $G \le_{S,T} H$.

Since, for the hierarchy of Figure 1, if X is more powerful than Y, then the control structures in Y are restrictions of the control structures in X, condition (1) is trivially satisfied with S' = T' = 1. It is of course results that establish condition (2) that have the greatest novelty.

To place our results in historical perspective, we follow Ledgard and Marcotty [12] in distinguishing the following extremes in simulations among control structures:

1) G is *functionally* simulated by H (written $G \le_f H$) if, under identical interpretations, G and H compute the same function.
2) G is *very strongly* simulated by H (written $G \le_{vs} H$) if $G \le_{1,1} H$ and, if Φ is an embedding inducing $\le_{1,1}$, then the domain of Φ and the range of Φ are identical sets.

In [2] it is shown that for an arbitrary goto program G there exists a while program H such that $G \le_f H$, while in [10] it is shown that for some goto program G there does not exist a while program H such that $G \le_{vs} H$. Several other notions of simulation intermediate to $\le_f$ and $\le_{vs}$ have also been used to study the relative power of classes of control structures [1,3,11,15].

The connection between our relation $\le_{S,T}$ and these relations is:

1) $\leq_{S,T}$ is weaker than $\leq_{vs}$ since we allow both space and time to increase and do not require Φ to have identical range and domain;

2) $\leq_{S,T}$ is stronger than $\leq_{f}$ since we require that paths be preserved in a weak sense;

3) $\leq_{S,T}$ deals only with combinatorial aspects of program structure and thus we make no assumptions about adding program variables or extra predicates (as were made for example in [1,2,11]).

We thus claim that the hierarchy theorems presented in this section span the relations used in previous studies.

We will make use of the following definitions. For any directed graph G, let

$$N_{in}^{G}(\ell,x) = |\{y: d_G(y,x) \leq \ell\}|$$

$$N_{out}^{G}(\ell,x) = |\{y: d_G(x,y) \leq \ell\}|.$$

Lemma 6.1: Suppose that G $\leq_{S,T}$ G* and let x be a node in G. Then

1) For any copy x* of x, $N_{out}^{G}(\ell,x) \leq N_{out}^{G*}(T\ell,x*)$;

2) For some copy x* of x, $N_{in}^{G}(1,x) \leq S \cdot N_{in}^{G}(T,x*)$.

The proofs of both (1) and (2) follow easily from the definition of $\leq_{S,T}$ and are left to the reader.

Theorem 6.2: do forever > while.

Proof: Let S,T be such that for all do forever programs G there is a while program H for which G $\leq_{S,T}$ H. By Lemma 6.1 part (2), for any node x in G there exists a copy x* of x such that

$$N_{in}^{G}(1,x) \leq S \cdot N_{in}^{H}(T,x*).$$

But since H is a while program, nodes in H have at most one ancestor pointer to them, so that

$$N^H_{in}(T,x^*) \le 2^T.$$

Thus

$$N^G_{in}(1,x) \le S2^T$$

for any do forever program G and any node x in G. This is a contradiction, since the number of ancestor pointers to nodes in do forever programs can be unbounded. □

Theorem 6.3: label exit > do forever.

Proof: Let S,T be such that for any label exit program G there exists a do forever program H such that G $\le_{S,T}$ H. Consider the label exit graph $G^{(n)}$ defined as follows:

1) $V^{(n)} = \{x_1,\ldots,x_n\}$
2) $x_i \xrightarrow[s]{} x_{i+1}$ for all $1 \le i < n$
3) $x_n \xrightarrow[a]{} x_i$ for all $1 \le i < n$

(see Figure 11). Then by construction

$$N^{G^{(n)}}_{out}(1,x_n) = n-1.$$

Let $H^{(n)}$ be the corresponding do forever program. Then a node x* in $H^{(n)}$ has at most two sons and one ancestor, so that

$$N^{H^{(n)}}_{out}(T,x^*) \le 3^T$$

Thus by Lemma 6.1 part (1)

$$n-1 = N^G_{out}(1,x_n) \le N^H_{out}(T,x^*) \le 3^T$$

where x* is any copy of x_n in H, a contradiction, since n is unbounded. □

Theorem 6.4: For d ≥ 2, goto_d > label exit.

Proof: Notice that arrays are included in $\underset{\sim\sim\sim}{\text{goto}}_f$ for f ≥ 4. Thus, by Theorem 5.1, we have $\underset{\sim\sim\sim}{\text{goto}}_f \not\leq$ label exit (f ≥ 4), since label $\underset{\sim\sim}{\text{exit}}$ programs are ancestor trees. To complete the proof it is sufficient to note that for any array G_n there is some H in $\underset{\sim\sim\sim}{\text{goto}}_2$ such that $G_n \leq_{4,1} H$. □

Theorem 6.5: $\underset{\sim\sim\sim}{\text{goto}}_\omega > \underset{\sim\sim\sim}{\text{goto}}_d$ for all d ≥ 1.

Proof: Let S,T be such that for all $\underset{\sim\sim\sim}{\text{goto}}_\omega$ programs G there exists a $\underset{\sim\sim\sim}{\text{goto}}_d$ program H such that $G \leq_{S,T} H$. By Lemma 6.1 part (1), for all nodes x in G there is a copy x* of x such that

$$N^H_{out}(T,x^*) \leq N^G_{out}(1,x).$$

But

$$N^H_{out}(T,x^*) \leq d^T,$$

since the out-degree of any vertex in H is at most d. This is a contradiction since in $\underset{\sim\sim\sim}{\text{goto}}_\omega$ $N^G_{out}(1,x)$ is unbounded. □

7. Conclusion

The methods for comparing data structures and control structures by the relation $\leq_{S,T}$ appear to be quite general, and there are several straightforward extensions of the data structure embedding results that recover relationships between other data structures.

In the case of control structures, the conclusions to be drawn are perhaps more widely varying and seem to give direction to further investigations. The observation in Corollary 5.4 that a standard schema construction is inherently inefficient leads us to question the status of other property-preserving transformations. We also believe that Theorem 6.5 has implications for the often quoted "theoretical foundations of structured programming"; this is particularly apparent because the goto program that enters the proof is itself a highly structured object. Indeed, a $\leq_{S,T}$ embedding into a label exit program fails, not because G_n is ill structured, but rather because of the densely hierarchical nature of the control flow. We thus offer programs of the form G_n as "structured" goto programs whose structures cannot be maintained by less general control structures. The exact relationship between ancestor trees and reducible flow graphs [7] is still unsettled, but some work has been done to place the reducible flow graphs in the hierarchy of Figure 1 [5]. More recent extensions of the results presented here give techniques for uncovering *total* space the time simulation trade-offs, as opposed to the worst-case analyses of this paper [5]. Finally, the extension of these results to parallel and asynchronous control structures appears to be possible and promises to yield important information about the relative power of non-sequential mechanisms.

Acknowledgements: The authors would like to thank W. Wesley Peterson for his comments on an earlier version of this paper and Arnold Rosenberg for a careful reading of the paper that led to a material improvement in presentation and for improvements to several proofs.

References

1. E. Ashcroft and Z. Manna.
The translation of GOTO programs to WHILE programs.
Proceedings of the IFIP 71 Congress, 250-255. North-Holland, 1972.

2. C. Böhm and G. Jacopini.
Flow-diagrams, Turing machines, and languages with only two formation rules.
Communications of the ACM 9(3):366-371, 1966.

3. J. Bruno and K. Steiglitz.
The expression of algorithms by charts.
Journal of the ACM 19(3):517-525, 1972.

4. O-J. Dahl et al.
Structured Programming.
Academic Press, 1972.

5. R. A. DeMillo, S. C. Eisenstat, and R. J. Lipton.
Space-time tradeoffs in structured programming.
Forthcoming.

6. E. Engeler.
Structure and meaning of elementary programs.
In E. Engeler, editor, Symposium on Semantics of Algorithmic Languages, 89-101.
Springer Lecture Notes in Mathematics 188, 1971.

7. M. S. Hecht and J. D. Ullman.
Characterizations of reducible flow graphs.
Journal of the ACM 21(3):367-375, 1974.

8. D. E. Knuth.
The Art of Computer Programming I: Fundamental Algorithms.
Addison-Wesley, 1968.

9. D. E. Knuth.
Structured programming with GOTO statements.
Stanford University Computer Science Department Report STAN-CS-74-416, 1974.

10. D. E. Knuth and R. W. Floyd.
Notes on avoiding "go to" statements.
Information Processing Letters 1:23-31, 1971.

11. S. R. Kosaraju.
Analysis of structured programs.
Journal of Computer and System Sciences 9(3):232-255, 1974.

12. H. F. Ledgard.
A genealogy of control structures.
University of Massachusetts Computer and Information Science Department Report, 1974.

13. C. B. Moler.
Matrix computations with FORTRAN and paging.
Communications of the ACM 15(4):268-270, 1972.

14. A. J. Perlis and C. Thornton.
Symbol manipulation by threaded lists.
Communications of the ACM 3(4):201-202, 1960.

15. W. W. Peterson, T. Kasami, and N. Tokura.
On the capabilities of the while, repeat, and exit statements.
Communications of the ACM 16(8):503-512, 1973.

16. A. L. Rosenberg.
Preserving proximity in arrays.
IBM Watson Research Center Report RC-4875, 1974.

$$\begin{tikzcd}
\text{computed goto or case} \arrow[d] \\
\text{goto with d way branching } (d \geq 2) \arrow[d] \\
\text{label exits} \arrow[d] \\
\text{do forever} \arrow[d] \\
\text{while}
\end{tikzcd}$$

Figure 1.

Note: The order is from most to least "powerful" with respect to the relation $\leq_{S,T}$.

$$\begin{tikzcd} x \arrow[r] & y \arrow[l, bend left] \end{tikzcd}$$

(a)

$$\begin{tikzcd} x \arrow[r, dash] & y \end{tikzcd}$$

(b)

Figure 2.

$$\begin{tikzcd}
(1,1) \arrow[r, dash] \arrow[d, dash] & (1,2) \arrow[r, dash] \arrow[d, dash] & (1,3) \arrow[d, dash] \\
(2,1) \arrow[r, dash] \arrow[d, dash] & (2,2) \arrow[r, dash] \arrow[d, dash] & (2,3) \arrow[d, dash] \\
(3,1) \arrow[r, dash] & (3,2) \arrow[r, dash] & (3,3)
\end{tikzcd}$$

Figure 3.

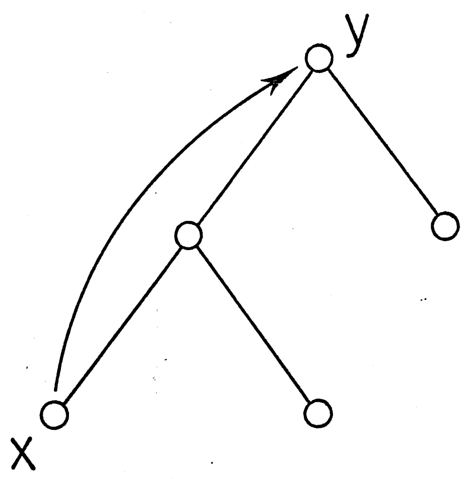

Figure 4.

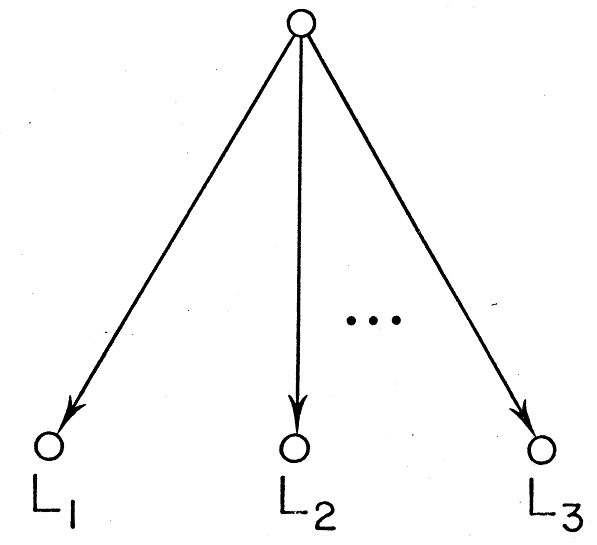

Figure 5.

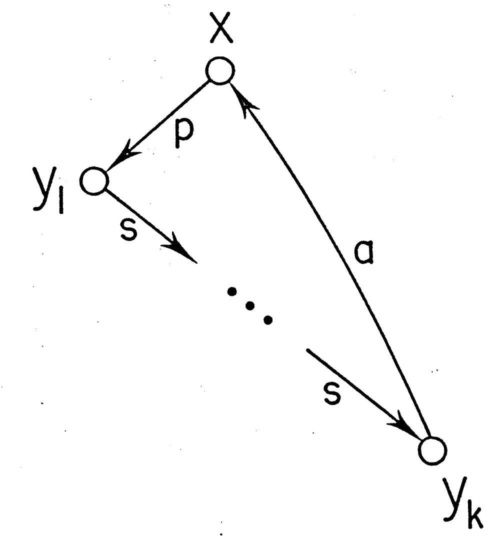

Figure 6a.

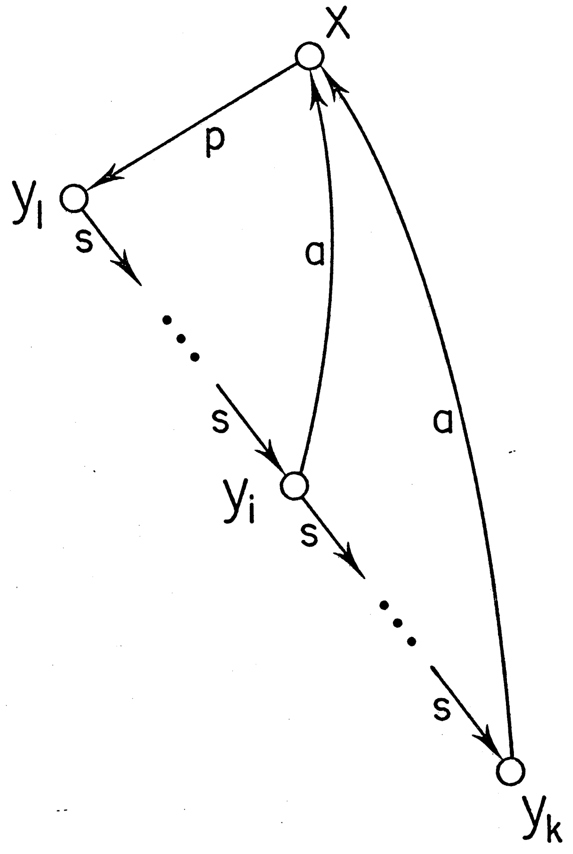

Figure 6b.

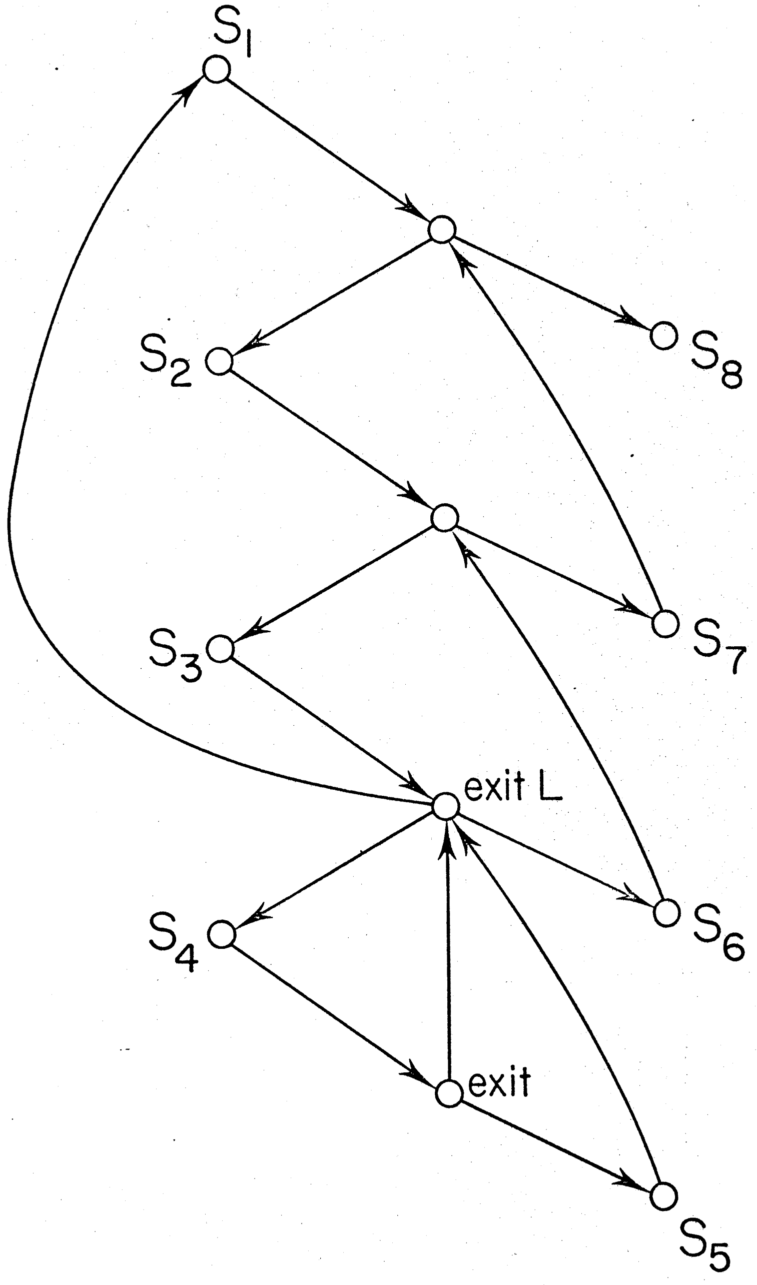

Figure 7.

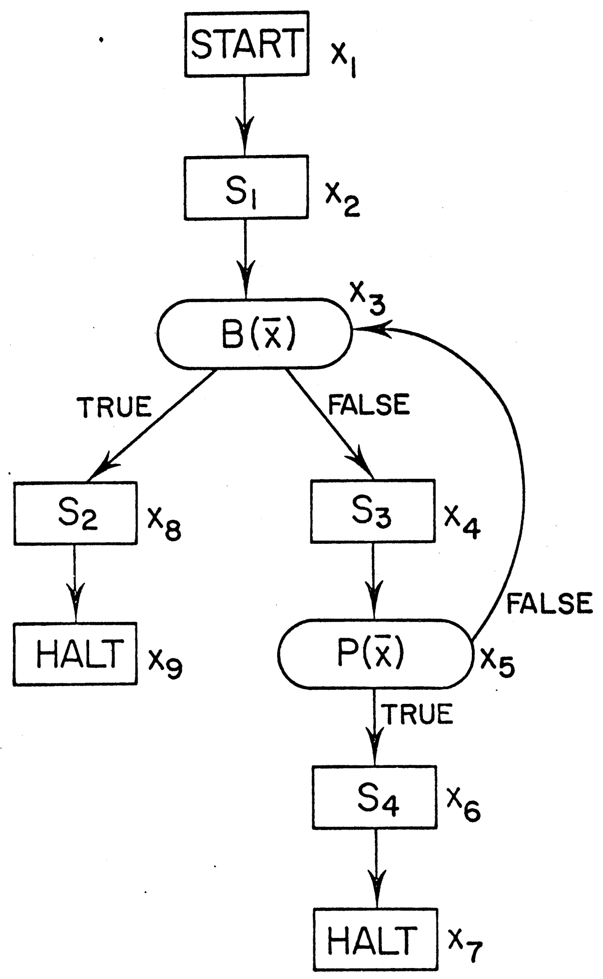

Figure 8a.

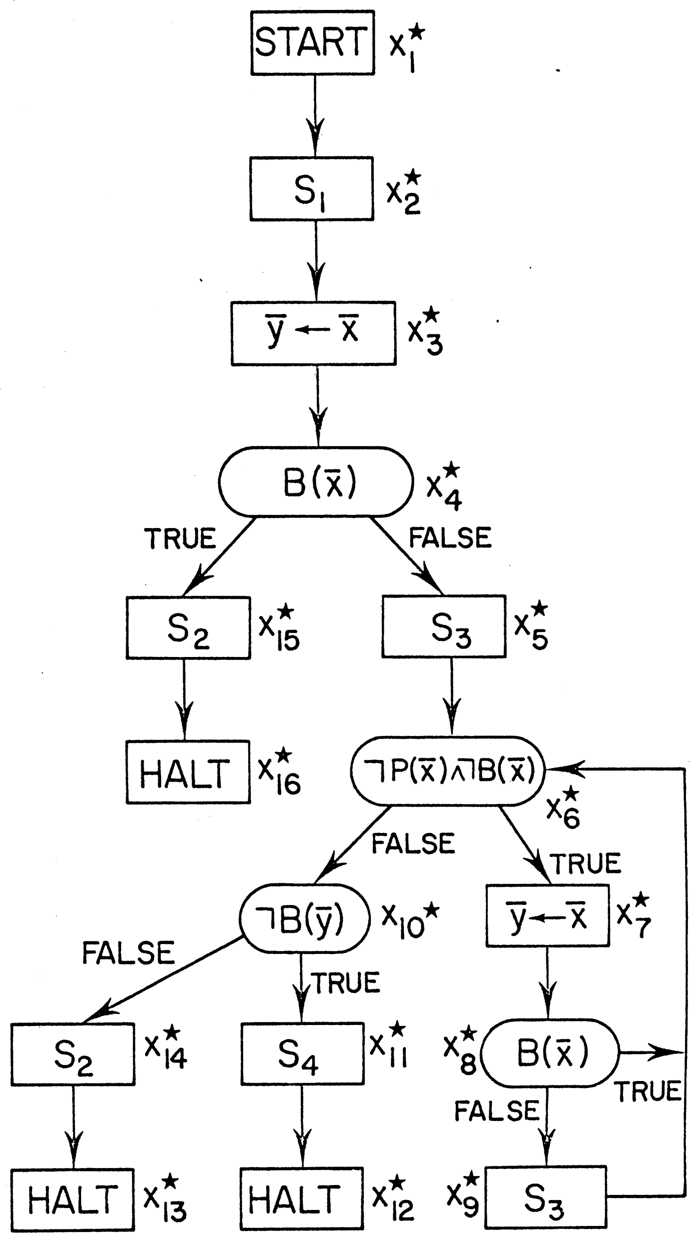

Figure 8b.

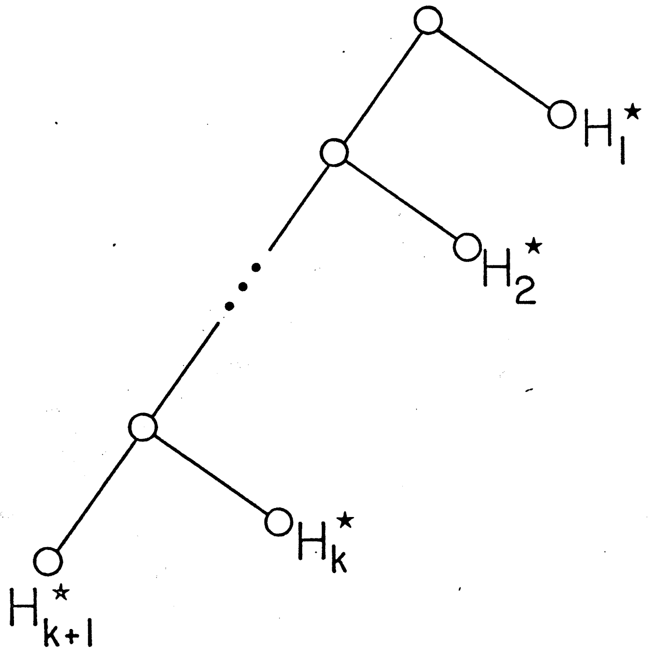

Figure 9.

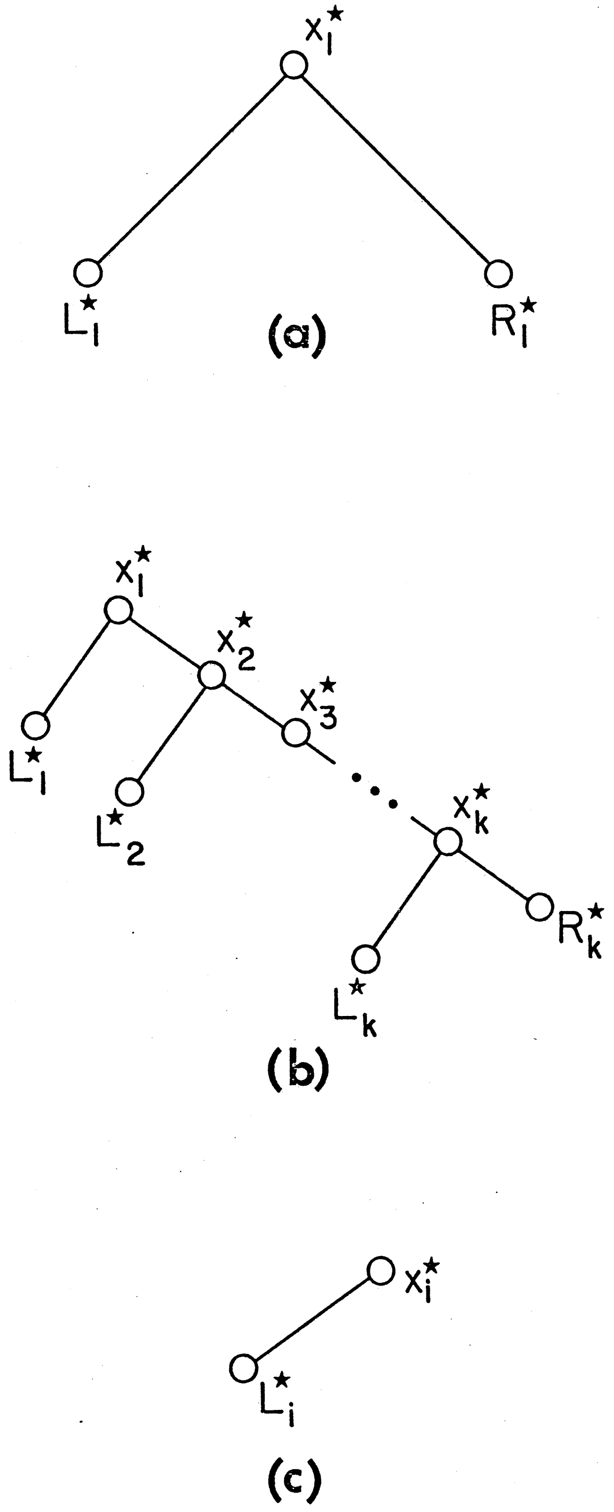

Figure 10.

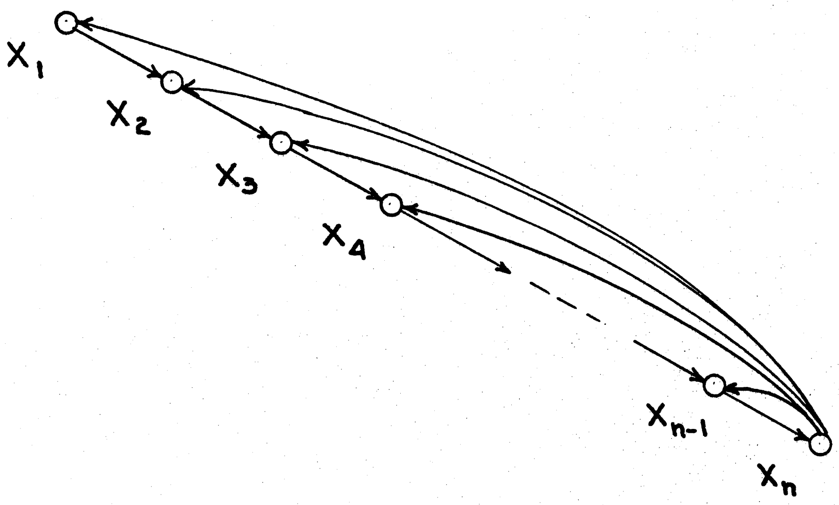

Figure 11.