INFORMATION AND CONTROL **49**, 10–51 (1981)

Parallel and Two-Way Automata on Directed Ordered Acyclic Graphs*

TSUTOMU KAMIMURA

Department of Computer Science, The University of Kansas, Lawrence, Kansas 66045

AND

GIORA SLUTZKI[†]

Department of Mathematics and Computer Science, Clarkson College of Technology, Potsdam, New York 13676

In this paper we study automata which work on directed ordered acyclic graphs, in particular those graphs, called derivation dags (*d*-dags), which model derivations of phrase-structure grammars. A rather complete characterization of the relative power of the following features of automata on *d*-dags is obtained: parallel versus sequential, deterministic versus nondeterministic and finite state versus a (restricted type of) pushdown store. New results concerning trees follows as special cases. Closure properties of classes of *d*-dag languages definable by various automata are studied for some basic operations. Characterization of general directed ordered acyclic graphs by these automata is also given.

1. INTRODUCTION

In this paper, we study automata which work on directed ordered acyclic graphs. In particular, we are interested in those graphs which model derivations of phrase-structure grammars. Such graphs are called derivation dags (*d*-dags).

D-dags are directed ordered acyclic graphs having (but not characterized by) the following properties: they are labeled, planar, connected and rooted.

* An extended abstract of this paper appeared in the proceedings of the Symposium on Mathematical Foundations of Computer Science, 1979, Olomouc, Czechoslovakia.

[†] Current address: Department of Computer Science, The University of Kansas, Lawrence, Kansas 66045.

0019-9958/81/040010–42$02.00/0

Figure 1 gives an example of a *d*-dag. The direction of all edges is downward. This *d*-dag has a root labeled by S and the two leaves are labeled respectively by a and b. It is easy to extract from this *d*-dag the following derivation of a phrase-structure grammar:

$$\underline{S} \Rightarrow \underline{A}B \Rightarrow BA\underline{B} \Rightarrow B\underline{AA}b \Rightarrow \underline{BC}b \Rightarrow ab,$$

where the underlined symbols indicate the substring being replaced at each step. Such a usage of graphs naturally extends the usage of derivation trees of context-free grammars.

Two distinct types of graph automata are studied as recognizers of *d*-dags. The first type, called parallel dag automaton, is an extended version of the ordinary tree automaton (Donor, 1970; Engelfriet, 1975a; Rounds, 1970; Thatcher and Wright, 1968). Historically, Arbib and Give'on (1968) were the first who extended the tree automaton to operate on directed ordered acyclic graphs. Their automaton was essentially a tree automaton and no particular problems concerning graph recognition were studied. Extensions of tree automata to graphs of derivations of phrase-structure grammars were

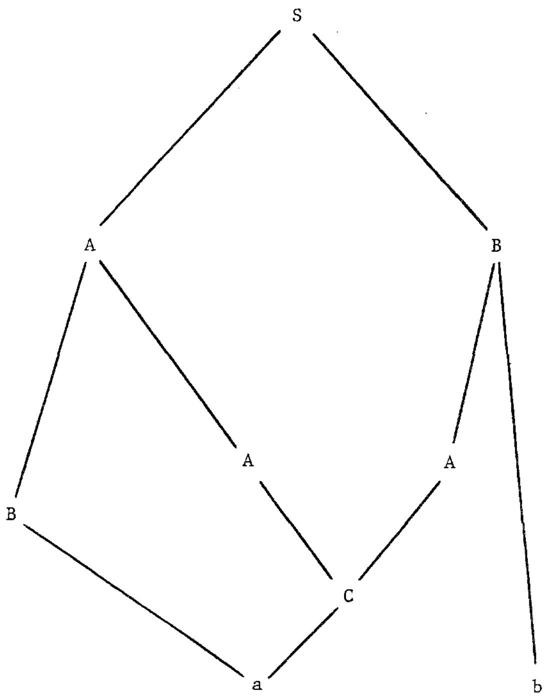


FIG. 1. An example of a *d*-dag.

studied by Buttelmann (1973) and Hart (1974, 1975a and b). Hart considered deterministic bottom-up (leaves-to-root) graph automata and Buttelmann studied the nondeterministic counterpart. We discuss both types of automata as well as the deterministic and nondeterministic top-down (root-to-leaves) automata. It is shown that deterministic and nondeterministic bottom-up parallel automata are equivalent for *d*-dags, whereas deterministic and nondeterministic top-down automata are inequivalent. It turned out that a large part of the theory of tree recognizers can be extended nicely to parallel dag automata on *d*-dags.

The second type of automaton is a two-way dag-walking automaton. This type of automaton appears in two rather different contexts. Blum and Hewitt (1967), Mylopoulos (1972) and Dilgram (1975) have used graph-walking automata in the context of syntactic pattern recognition. They use graphs to represent "patterns" and discuss various automata as a means to characterize different pattern classes. More relevant to our study are the papers of Aho and Ullman (1971) and Engelfriet *et al.* (1978) in the framework of syntax-directed translations and tree transducers. In fact, our two-day dag-walking automata originated by considering the recognizer version of the tree-to-string transducers studied in these papers. We introduce dag-walking automata and study the power of their deterministic and nondeterministic versions. It is shown that with pushdown storage, deterministic and nondeterministic automata of this type are equivalent whereas without the pushdown store the nondeterministic automaton is more powerful than the deterministic one. It is also shown that automata with the pushdown facility are more powerful than those without it. These results on two-way dag-walking automata hold also for the tree case and they are new.

Then we compare parallel dag automata and two-way dag-walking automata. We show that the deterministic bottom-up parallel dag automata and the two-way dag-walking automata with pushdown are equipotent in their power of recognizing *d*-dags. Our results on comparison of the power to recognize *d*-dags by various automata introduced in this paper is summarized in Fig. 22.

As a result of comparion of dag automata, we obtain four different classes of *d*-dag languages. We then study the closure properties of these classes under boolean operations and finite state relabelings, which are the transducer versions of parallel dag automata. Closure of deterministic finite state dag-walking automata under union and complement remains open.

Finally we show that the results in Fig. 22 hold also for general directed ordered acyclic graphs (DOAGs) with the exception of the relation between deterministic and nondeterministic bottom-up parallel automata. The inclusion diagram of the classes of DOAG languages defined by various dag automata is given by Fig. 24.

2. Definitions of the Graphs

The graphs which we intend to study are subclasses of the directed acyclic and ordered graphs. In this section, we formally define these objects. We now begin with notational preliminaries.

A *directed ordered graph* G is a construct (N, E, H), where N is a set of *nodes*, $E \subseteq N \times N$ is a set of *edges* and H is a partial order on N satisfying: $(x, y) \in H$ or $(y, x) \in H$ if there is a node $z \in H$ such that $((x, z) \in E$ and $(y, z) \in E)$ or $((z, x) \in E$ and $(z, y) \in E)$. Thus, for each node $x \in N$, the set of its *fathers* $\{y \mid (y, x) \in E\}$ and the set of *sons* $\{y \mid (x, y) \in E\}$ are totally ordered by H. A *path* of length n, $n \geqslant 0$, from x to y in G is a sequence $\langle v_0, v_1, ..., v_n \rangle$ of nodes $v_i \in N$, such that $v_0 = x$, $v_n = y$ and $(v_i, v_{i+1}) \in E$ for $0 \leqslant i < n$. A directed ordered graph G is *acyclic* if there is no path of length more than 0 from a node to itself. G is *connected* if for any two distinct nodes x and y, there is a sequence of nodes $\langle v_1, ..., v_n \rangle$ such that $v_1 = x$, $v_n = y$ and $((v_i, v_{i+1}) \in E$ or $(v_{i+1}, v_i) \in E)$ for $1 \leqslant i < n$. For a node $x \in N$, the *in-degree* of x is the number of nodes y such that $(y, x) \in E$. Similarly, the *out-degree* of x is the number of nodes y such that $(x, y) \in E$. A *root* is a node with in-degree 0 and a *leaf* is a node with out-degree 0. A directed ordered graph is *rooted* if it has only one root. A picture of a directed

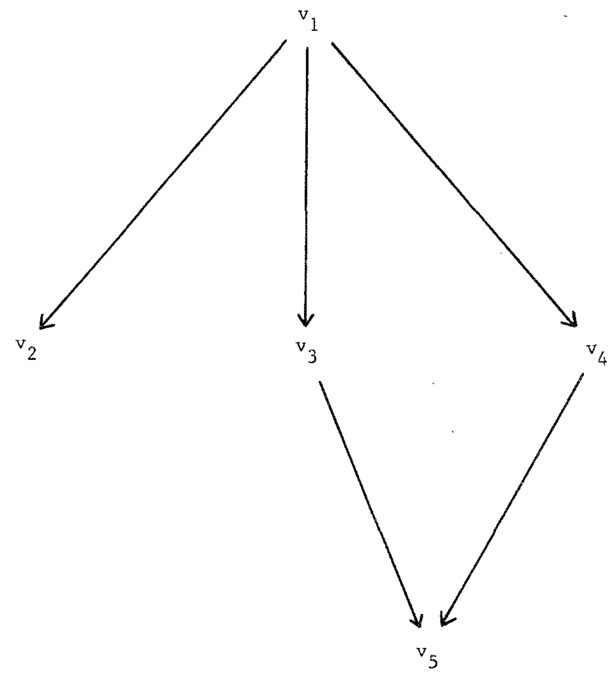


Fig. 2. Directed ordered graph.

ordered graph is drawn in the standard manner. For example. Fig. 2 shows a directed ordered graph G with $N = \{v_1, v_2, v_3, v_4, v_5\}$, $E = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_3, v_5), (v_4, v_5)\}$, and $H = \{(v_1, v_1), (v_2, v_2), (v_3, v_3), (v_4, v_4), (v_5, v_5), (v_2, v_3), (v_2, v_4), (v_3, v_4)\}$. This graph is rooted, connected and acyclic. A directed ordered graph is *planar* if it can be drawn in the plane in such a way that its edges intersect only at the nodes of the graph. A directed ordered graph is *labeled* if every node is labeled by an element of some alphabet.

A directed ordered graph is a *tree* if it is rooted, connected, acyclic and planar, and the maximal in-degree of nodes is at most 1. Figure 3 shows a picture of a labeled tree. The direction of edges is downward. The *height* of a tree is the number of edges contained in the longest path in the tree.

We now introduce two types of directed ordered labeled graphs. The labels are taken from an alphabet where the in-degree and out-degree of a node are used as rank numbers for the possible labels of that node. A *doubly ranked alphabet* is a set $\Sigma = \bigcup_{i,j} \Sigma_{ij}$, where each Σ_{ij} is a finite set and only for a finite number of i and j, $\Sigma_{ij} \neq \varnothing$. An element $\sigma \in \Sigma_{ij}$ is said to have *head-rank* i and *tail-rank* j. We also define $\Sigma_{*j} = \bigcup_i \Sigma_{ij}$ and $\Sigma_{i*} = \bigcup_j \Sigma_{ij}$. When each symbol in Σ has head-rank 0 or 1, Σ is called a *ranked alphabet* (used to label trees).

2.1. Definition. Let Σ be a doubly ranked alphabet. A DOAG d over Σ is a directed ordered graph which is acyclic, connected, rooted and labeled in such a way that a node having i fathers and j sons in d has a label

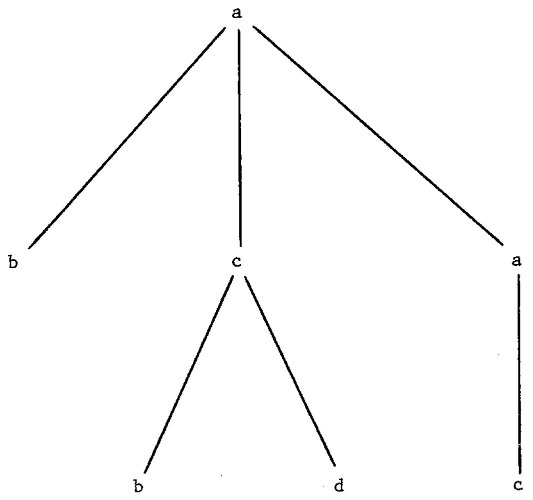


Fig. 3. A labeled tree.

$\sigma \in \Sigma_{ij}$, $i,j \geqslant 0$. The set of all DOAGs over Σ is denoted as G_Σ. Any subset of G_Σ is a *DOAG language*. ∎

Derivation-dags (*d*-dags) are special DOAGs which model derivations of phrase-structure grammars, and they are the main objects of our study. They are defined inductively as follows.

2.2. DEFINITION. Let Σ be a doubly ranked alphabet. The set of *partial d-dags over Σ*, denoted by P_Σ, is defined inductively as follows.

(i) If $a \in \Sigma_{0*}$ then $a \in P_\Sigma$; $\text{root}(a) = a$, $\text{leaves}(a) = a$ (when convenient we will identify a node with its label).

(ii) Let $d \in P_\Sigma$ with $\text{leaves}(d) = a_1 a_2 \cdots a_n$ and let $a_i \in \Sigma_{*m}$, $m \geqslant 1$; let $b_1, b_2, ..., b_m \in \Sigma_{1*}$. We add to d m new nodes labeled by $b_1, ..., b_m$ in this order and m new edges (a_i, b_j), $1 \leqslant j \leqslant m$. Then d' in Fig. 4(a) is in P_Σ with $\text{root}(d') = \text{root}(d)$ and $\text{leaves}(d') = a_1 \cdots a_{i-1} b_1 \cdots b_m a_{i+1} \cdots a_n$.

(iii) Let $d \in P_\Sigma$ with $\text{leaves}(d) = a_1 a_2 \cdots a_n$ and $a_i, a_{i+1}, ..., a_j \in \Sigma_{*1}$ for some $1 \leqslant i \leqslant j \leqslant n$. Let $b \in \Sigma_{(j-i+1)*}$. We add to d a new node labeled by b and $j - i + 1$ new edges (a_k, b), $i \leqslant k \leqslant j$; then d' of Figure 4(b) is in P_Σ with $\text{root}(d') = \text{root}(d)$ and $\text{leaves}(d') = a_1 \cdots a_{i-1} b a_{j+1} \cdots a_n$. ∎

The set of *d-dags over Σ* is then

$$D_\Sigma = \{d \in P_\Sigma \mid \text{leaves}(d) \in \Sigma_{*0}^*\},$$

where the superscript * denotes the Kleene closure operation.

For a ranked alphabet Σ, *d*-dags become trees, and the set of all trees over

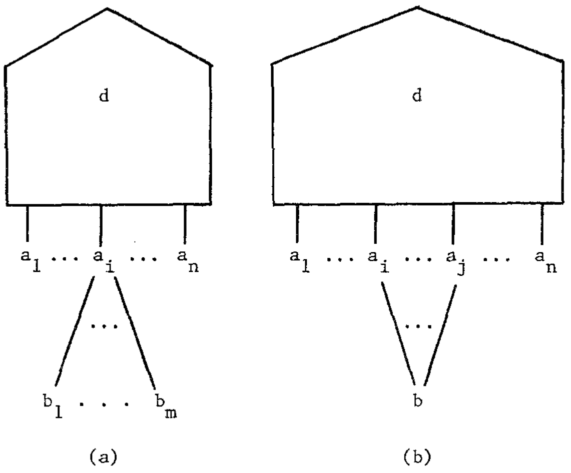

d
$a_1 \cdots a_i \cdots a_n$
$b_1 \cdots b_m$
(a)
d
$a_1 \cdots a_i \cdots a_j \cdots a_n$
b
(b)

FIG. 4. Construction rules for *d*-dags.

Σ is denoted by T_Σ. Any subset of $D_\Sigma(T_\Sigma)$ is called a *d-dag* (*tree*) *language*. The operation of taking the leaves (of a *d*-dag) is extended to *d*-dag languages and to classes of *d*-dag languages as follows.

For $L \subseteq D_\Sigma$ $\text{leaves}(L) = \{\text{leaves}(d) \mid d \in L\}$ and for a class of *d*-dag languages $\mathscr{L}$, $\text{leaves}(\mathscr{L}) = \{\text{leaves}(L) \mid L \in \mathscr{L}\}$. In the tree case, the operation leaves is the same as "yield" operation in (Baker, 1973; Engelfriet, 1975a; Engelfriet *et al.*, 1978).

Remarks. (i) *D*-dag is, as remarked earlier, an ordered graph and this order is expressed as a left to right order among the direct predecessors and direct successors of a node in the way we draw the graph.

(ii) The direction of all edges is downward, and therefore no cycles are introduced.

(iii) Both connectivity and planarity are preserved by the construction schemes of Definition 2.2.

(iv) The definition excludes the possibility of a subgraph of Fig. 5(a) which does not have any immediately evident derivation meaning. On the other hand, the definition allows a subgraph of Fig. 5(b) with which an obvious derivation meaning can be associated. In Hart (1974, 1975a and b), both types of subgraphs were excluded.

(v) A *d*-dag has a unique root. ▮

2.3. Example. Let $\Sigma = \{a, b, c, e\}$ be a doubly ranked alphabet with: $\Sigma_{03} = \{a\}$, $\Sigma_{11} = \{b, c, e\}$, $\Sigma_{12} = \{a, c\}$, $\Sigma_{21} = \{b\}$ and $\Sigma_{20} = \{e\}$. Then the labeled graph d of Fig. 6 is a *d*-dag with $\text{root}(d) = a$ and $\text{leaves}(d) = ee$. d describes the process of the derivation of the rewriting system with rules: $a \to bca$, $c \to ce$, $a \to cc$, $bc \to b$, $be \to e$ and $cc \to e$. ▮

Phrase-structure grammars whose derivations are modeled by *d*-dags are restricted grammars in the sense that right-hand sides of their non-context-

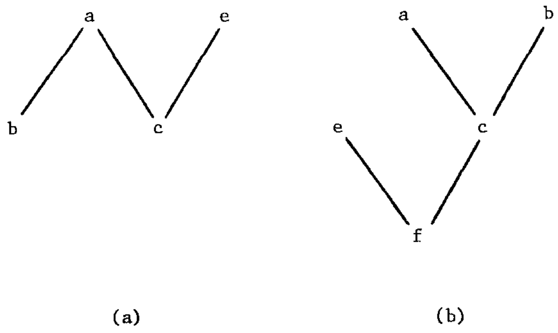


Fig. 5. Subgraph structures.

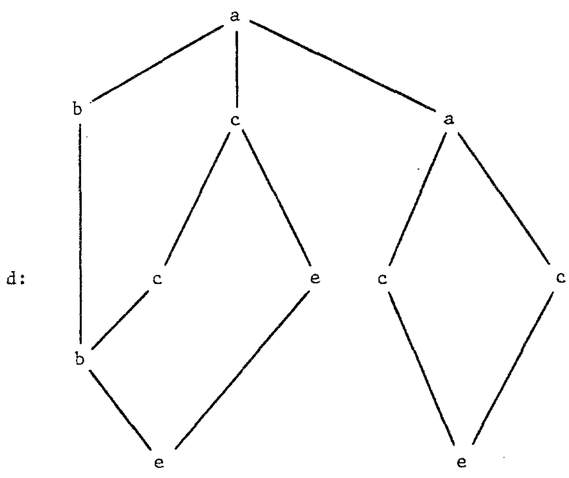


FIG. 6. An example of a d-dag.

free rules consist of a single letter. However, such grammars are sufficiently general in that for each phrase-structure grammar, there is an equivalent restricted grammar in this sense which has the same "derivation structure" as the given grammar.

3. PARALLEL DAG AUTOMATA

Finite one-way automata operating on d-dags are studied, one-way meaning either downward (top-down) or upward (bottom-up). These automata traverse (process) the d-dag in a parallel fashion in that a dynamically changing number of "copies" of the finite (control of the) automaton simultaniously point into the d-dag, one pointer per copy. The pointers move on the d-dag by a "merge-split" operation in which copies of the automaton on edges entering a node, n say, die, but simultaneously give birth to new copies on edges going out of that node. In this way the pointers have moved past n (this was the top-down situation; the bottom-up case is just the converse).

For a doubly ranked alphabet Σ we define a companion alphabet $\Sigma' = \{\sigma' \mid \sigma \in \Sigma\}$ such that σ and σ' have precisely the same head and tail ranks. Formal definition of parallel dag automata follows.

3.1. DEFINITION. A *finite dag automaton* is a construct $A = (Q, \Sigma, R)$, where Q is a finite set of *states*, Σ is a doubly ranked alphabet and R is a finite set of *rules* of the form $r: \alpha \to \beta$. α and β are respectively the left-hand side and right-hand side of r. A is *deterministic* if two different rules have different left-hand sides; otherwise A is *nondeterministic*. A being *top-down* or *bottom-up* depends on the form of α and β above, as follows.

(a) A is top-down if the rules in R are of the form

$$[p_1 p_2 \cdots p_n]\,\sigma \to \sigma'(q_1 q_2 \cdots q_m)$$

with $\sigma \in \Sigma_{nm}$, $p_1,\ldots,p_n$, $q_1,\ldots,q_m \in Q$.

(b) A is bottom-up if the rules in R are of the form

$$\sigma(q_1 q_2 \cdots q_m) \to [p_1 p_2 \cdots p_n]\,\sigma'$$

with $\sigma \in \Sigma_{nm}$, $p_1,\ldots,p_n$, $q_1,\ldots,q_m \in Q$. ▮

In the above definition the symbols on the right-hand sides of rules get primed only to signify that σ has been processed and to prevent the appearance of redundant copies of the automaton on the d-dag (see below).

Let $A = (Q, \Sigma, R)$ be a dag automaton. A *configuration* of A is a d-dag over the doubly ranked alphabet $\Delta = \Sigma \cup \Sigma' \cup Q$, where $Q \subseteq \Delta_{11}$; for any two occurrences of states in a configuration neither of them is below the

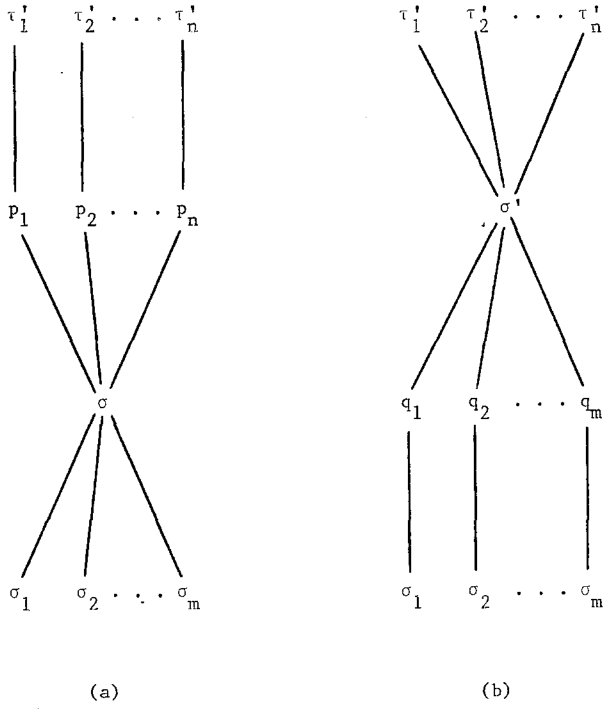

τ'_1 τ'_2 . . . τ'_n
p_1 p_2 . . . p_n
σ
σ_1 σ_2 . . . σ_m
(a)
τ'_1 τ'_2 . . . τ'_n
σ'
q_1 q_2 . . . q_m
σ_1 σ_2 . . . σ_m
(b)

FIG. 7. Relation $\vdash_A$ of top-down automaton.

other, and every path from a primed node (i.e., a node whose label is primed) to an unprimed node (or vice versa) must pass through a node labeled by an element of Q. We define a relation $\vdash_A$ between configurations of A as follows.

(i) If A is top-down, d_1, d_2 are configurations of A, then $d_1 \vdash_A d_2$ if d_1 contains a subgraph of Fig. 7(a), R has a rule $[p_1 p_2 \cdots p_n]\, \sigma \to \sigma'(q_1 q_2 \cdots q_m)$ and d_2 is obtained from d_1 by replacing the subgraph of Fig. 7(a) by the subgraph of Fig. 7(b).

(ii) If A is bottom-up, d_1, d_2 are configurations of A then $d_1 \vdash_A d_2$ if d_1 contains a subgraph of Fig. 8(a), R has a rule $\sigma(q_1 q_2 \cdots q_m) \to [p_1 p_2 \cdots p_n]\, \sigma'$ and d_2 is obtained from d_1 by replacing the subgraph of Fig. 8(a) by the subgraph of Fig. 8(b).

Remark. This definition of $\vdash_A$ includes the case $n = 0$ or $m = 0$ or both. For instance, if a top-down automaton A has a rule $\sigma \to \sigma'(q_1 \cdots q_m)$ in R, subgraphs of (a) and (b) of Fig. 7 become subgraphs of (a) and (b) of Fig. 9, respectively. With this type of rule, A can start its computation at the root of an input d-dag. For a rule $[p_1 \cdots p_n]\, \sigma \to \sigma'$, those two subgroups become

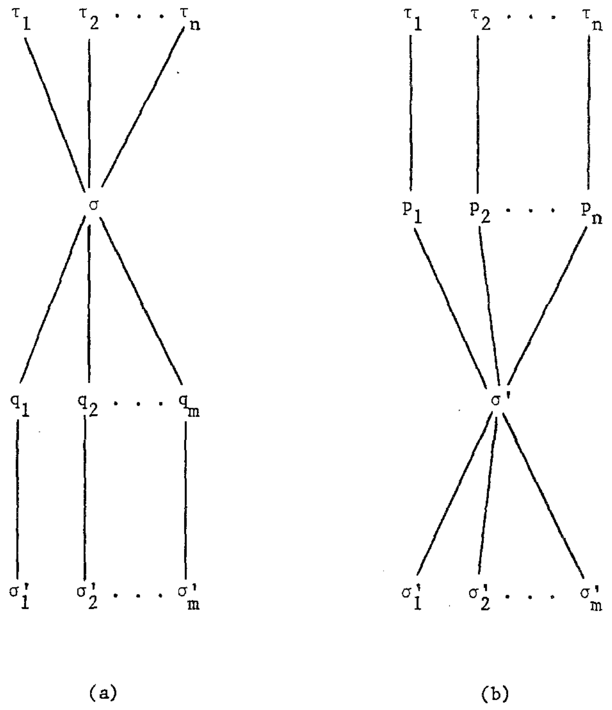


FIG. 8. Relation $\vdash_A$ of a bottom-up automaton.

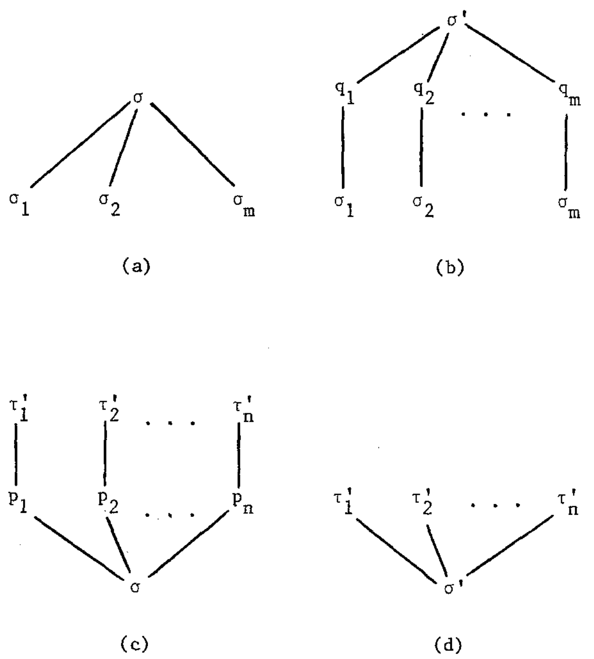


FIG. 9. Relation $\vdash_A$ of top-down automaton for $n = 0$ or $m = 0$.

subgraphs of (c) and (d), respectively, in Fig. 9. A can successfully terminate its computation on an input by applying this type of rule at each leaf of the input. In particular, for $n = m = 0$, $\sigma \vdash_A \sigma'$ iff $\sigma \to \sigma'$ is a rule of A. The case for a bottom-up automaton is analogous. ∎

Given $\vdash_A$, $\vdash_A^*$ is the reflexive–transitive closure of $\vdash_A$. For any d-dag $d \in D_\Sigma$, let $d' \in D_{\Sigma}$, be the d-dag obtained by priming all the labels in d. The *d-dag language* accepted by the dag automaton $A = (Q, \Sigma, R)$ is $L(A) = \{d \in D_\Sigma \mid d \vdash_A^* d'\}$.

NT and *DT* will denote respectively the sets of all nondeterministic and deterministic top-down dag automata and similarly *NB* and *DB* in the bottom-up case. For a class K of dag automata, $\mathscr{L}(K) = \{L(A) \mid A \in K\}$ is the class of d-dag languages defined by automata in K.

Remarks. (i) These definitions are actually not restricted to recognition of d-dags; they are also valid for DOAGs. It is with this reason that we use "dag" automaton instead of "d-dag" automaton. In Arbib and Give'on, 1968, bottom-up DOAG automata were discussed for slightly different DOAGs in which fathers of each node are not ordered; hence their automata correspond to the case $p_1 = p_2 = \cdots p_n$, in (b) of Fig. 8. Recognition of DOAGs by dag automata will be studied in Section 8.

(ii) The conventional tree automata are defined in a slightly different manner (see Engelfriet, 1975a; Thatcher, 1973). For instance, a bottom-up tree automaton arrives at each node of (tail-) rank m with a sequence of m states (one state for each direct subtree of the node), and the transition function of the automaton determines the state at this node using the sequence of states and the symbol labeling the node. It accepts an input tree if and only if the automaton is in one of the specified final states at a root of the tree. It is easy to see that in the case of ranked alphabet, bottom-up dag automata can precisely describe such tree automata and vice versa. The same holds for the top-down case. Thus, each type of the dag automata reduces to the corresponding type of tree automata for a ranked alphabet. ∎

3.2. EXAMPLE. Let $\Sigma_{02} = \Sigma_{12} = \{a\}$, $\Sigma_{11} = \{b\}$ and $\Sigma_{20} = \{c\}$. Consider the top-down dag automaton $A = (\{p, q\}, \Sigma, \{a \to a'(pp), a \to a'(pq), a \to a'(qp), a \to a'(qq), [p]\, a \to a'(pp), [p]\, a \to a'(pq), [p]\, a \to a'(qp),$

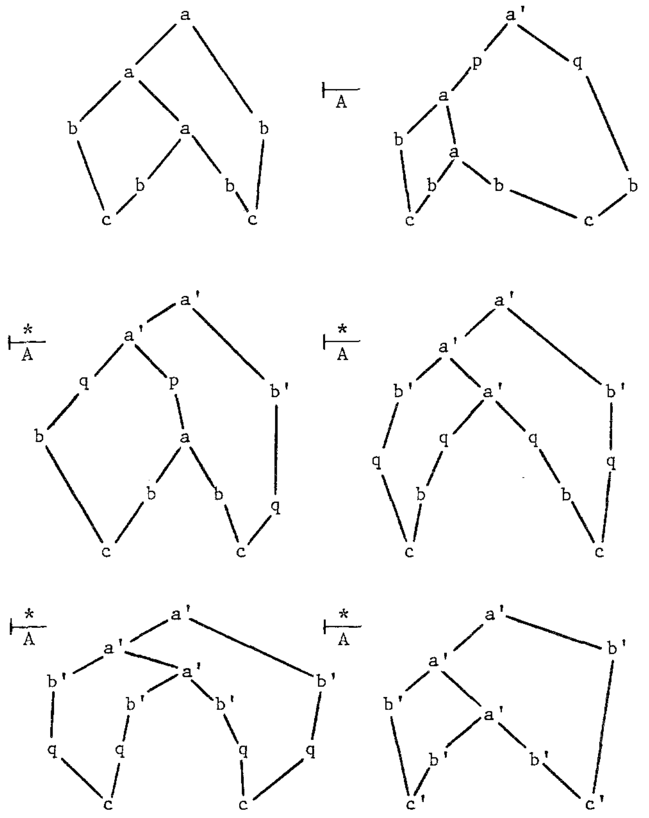


FIG. 10. An example of a computation of A.

$[p]\,a \to a'(qq)$, $[q]\,b \to b'(q)$, $[qq]\,c \to c'\})$. Figure 10 illustrates a computation of A. The automaton recognizes the language $L = \{d \in D_\Sigma \mid \text{every path in } d \text{ from the root to a leaf is in } a^\dagger b^* c\}$. ▮

3.3. EXAMPLE. Let $\Sigma_{11} = \{a, b\}$ and $\Sigma_{02} = \Sigma_{22} = \Sigma_{20} = \{c\}$. Consider the following d-dag language $L = \{d_n \mid n \geqslant 1\}$ over Σ, where d_n is the d-dag of Fig. 11. The language L is recognized by the deterministic bottom-up dag automaton $A = (\{p, p', q, q'\}, \Sigma, R)$, where

$$R = \{c \to [pq]\,c', a(p) \to [p']\,a', b(q) \to [q']\,b', c(p'q') \to [pq]\,c, c(p'q') \to c'\}. \quad ▮$$

We are now ready to compare the machines defined above. First we show that nondeterministically there is no difference between bottom-up and top-down recognition.

3.4. THEOREM. $\mathscr{L}(NT) = \mathscr{L}(NB)$.

Proof. Define $A_1 = (Q, \Sigma, R_1) \in NT$ and $A_2 = (Q, \Sigma, R_2) \in NB$ to be associated with each other if they satisfy:

$$[p_1 \cdots p_n]\,\sigma \to \sigma'(q_1 \cdots q_m) \in R_1$$
$$\text{if and only if } \sigma(q_1 \cdots q_m) \to [p_1 \cdots p_n]\,\sigma' \in R_2.$$

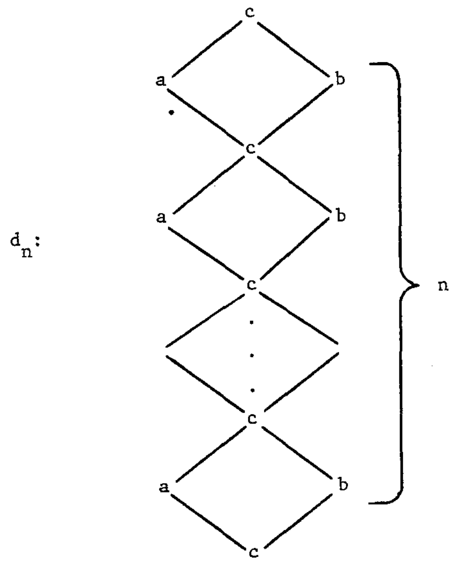


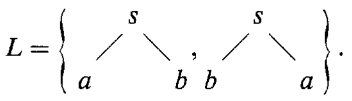
FIG. 11. A dag d_n.

By a strightforward induction on the number of steps of computation of A_1 and A_2, $d_1 \vdash_{A_1}^{*} d_2$ iff $\hat{d}_2 \vdash_{A_2}^{*} \hat{d}_1$, where $\hat{d}_i$ is obtained from d_i by exchanging labels σ and σ' for every $\sigma \in \Sigma$, $i = 1$ or 2. Hence, $L(A_1) = L(A_2)$. Obviously each automaton of one type can be defined in terms of the other type, concluding the proof. ▮

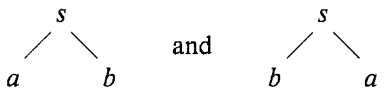
In the tree case, it is known (for example, see Engelfriet, 1975a) that, with respect to recognition, the deterministic top-down automaton is less powerful than its nondeterministic version. Since our various dag automata operate like the corresponding tree automata for tree inputs, the next result is immediate.

3.5. THEOREM. $\mathscr{L}(DT) \subsetneq \mathscr{L}(NT)$.

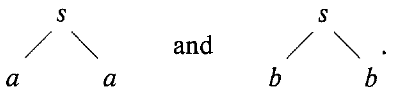
Proof. Let $\Sigma_{10} = \{a, b\}$ and $\Sigma_{02} = \{s\}$. Consider the (tree) language



Then any deterministic top-down dag automaton accepting



must also accept



This shows that $L \notin \mathscr{L}(DT)$, while obviously $L \in \mathscr{L}(NT)$. ▮

The next theorem shows that the deterministic and nondeterministc bottom-up automata are equally powerful. In the tree case it is easily proved by a standard subset construction. However, in the d-dag case, a simple-minded subset construction does not work, because of the "non-tree-like" rules $\sigma(q_1 \cdots q_m) \rightarrow [p_1 \cdots p_n]\, \sigma'$ with $n > 1$. The difficulty is the following. Let $A = (Q, \Sigma, R)$ be a nondeterministic bottom-up dag automaton. In an input d-dag d, consider the node σ with two adjacent fathers δ_1 and δ_2.

Define two sequences of nodes: $x_1 = \delta_1, x_2, \ldots$; $y_1 = \delta_2, y_2, \ldots$ such that $x_{i+1}(y_{i+1})$ is the rightmost (leftmost) father of $x_i(y_i)$ for $i > 0$ in d. Because of rootedness of d, these two sequences have eventually common nodes and let τ be the first node which appears in both sequences; since d is planar, τ is the nearest common ancestor of δ_1 and δ_2. The relation of δ_1, δ_2 and τ is illustrated in Fig. 12. This subgraph structure is called a *window* in d. This window is the *right-window* (*left-window*) of an edge in the path from τ to σ through $\delta_1(\delta_2)$. Now assume that A has two possibilities to proceed from σ to δ_1 and δ_2: one with $p_1 p_2$ and the other with $q_1 q_2$; see the figure. Then let p'_1 and p'_2 (q'_1 and q'_2) be a possible pair of states in which A arrives at the node τ if p_1 and p_2 (q_1 and q_2) are the states in which A leaves σ. Now suppose A has a rule with $\tau(p'_1 q'_2)$ as its left-hand side. Since the automaton defined by the subset construction does not keep any information on the rules used and simulates A by considering all the possible combinations of states, it may apply the above rule at τ and thus undesired d-dags may be accepted.

We modify the subset construction to overcome the difficulty as follows. Let $\hat{A} \in DB$ be the dag automaton to be constructed. A state of $\hat{A}$ is a set of triples (q, ℓ, r), where q is a state of A, and ℓ and r are rules of A. For a state P of $\hat{A}$, and $(q, \ell, r) \in P$, (a copy of) $\hat{A}$ being in state P at an edge between node τ and its ith son (while operating on an input d-dag d) means (roughly) that there is a corresponding computation of A on d in which (a copy of) A

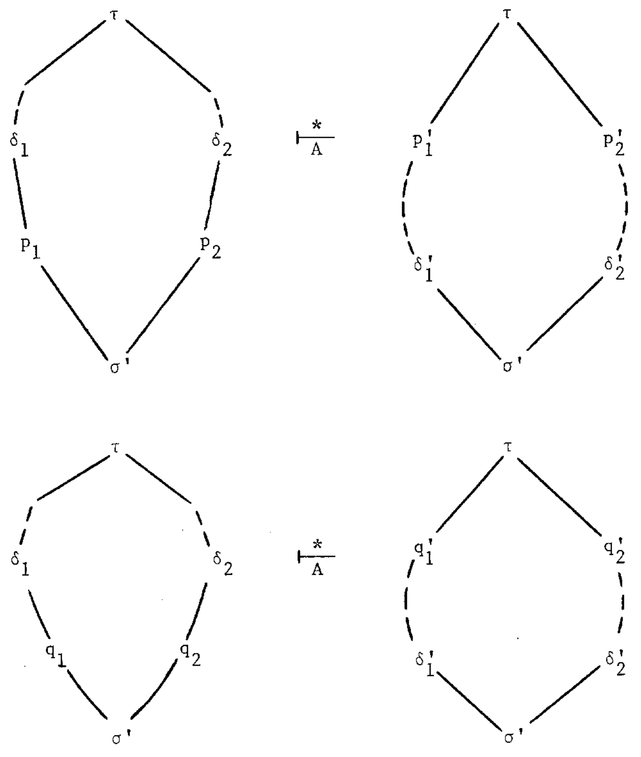


FIG. 12. The difficulty of subset construction.

arrives at that edge in state q such that $\ell(r)$ was the rule of A used at the bottom-node $\sigma_1(\sigma_2)$ of the left-(right-)window of the edge, see Fig. 13. In the case that the left-(right-)window does not exist, $\ell(r)$ is not needed and it is just a dummy symbol. In this way $\hat{A}$ remembers the states of A as well as the rules used (to produce such states) at the bottom node of each window in an input d-dag. Then, at a top node of the window, $\hat{A}$ uses this information to find out the correct combinations of states of A to apply its rules. For instance, in Fig. 12 $\hat{A}$ does not apply the rule with $\tau(p'_1 q'_2)$ as its left-hand side at τ, because the rule r associated to p'_1 and the rule ℓ associated to q'_2 are different. Note that properties of rootedness and planarity of d-dags are crucial here.

3.6. THEOREM. $\mathscr{L}(DB) = \mathscr{L}(NB)$.

Proof. We have to show $\mathscr{L}(NB) \subseteq \mathscr{L}(DB)$. Let $A = (Q, \Sigma, R)$ be a nondeterministic bottom-up dag automaton. We construct $\hat{A} = (\hat{Q}, \Sigma, \hat{R})$ in DB as follows. The set of states is $\hat{Q} = \mathscr{P}(\{(q, \ell, r) \mid q \in Q$ and $\ell, r \in R \cup \{NIL\}\})$, where NIL is a dummy symbol and $\mathscr{P}(\cdots)$ denotes the power set operation. Let $\sigma \in \Sigma_{nm}$ and $Q_1, \ldots, Q_m \in \hat{Q}$. We explain the

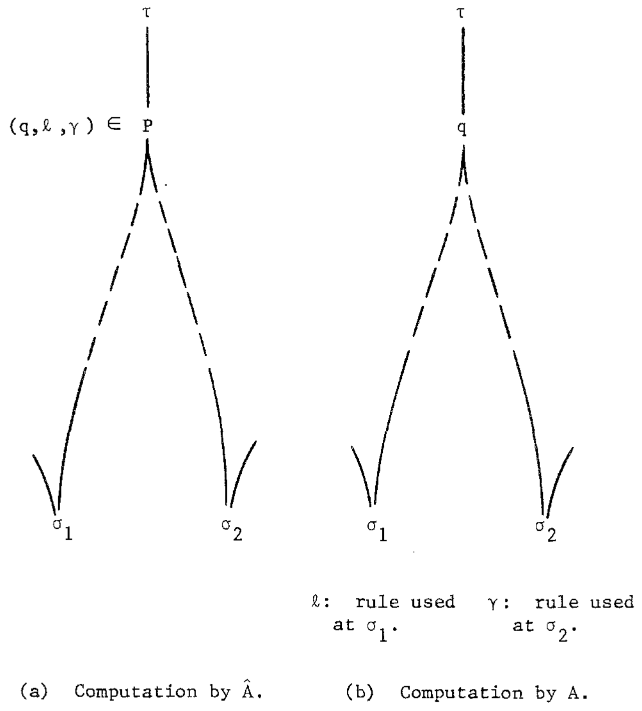


FIG. 13. Computation of A and $\hat{A}$.

construction of $P_1, \dots, P_n \in Q$ such that $\sigma(Q_1 \cdots Q_m) \to [P_1 \cdots P_n]\,\sigma'$ will be in $\hat{R}$, the set of rules of A.

Let $(q_i, \ell_i, r_i) \in Q_i$ $(1 \leqslant i \leqslant m)$ such that $r_i = \ell_{i+1}$ $(1 \leqslant i \leqslant m)$ and such that $s\colon \sigma(q_1 \cdots q_m) \to [p_1 \cdots p_n]\,\sigma'$ is in R $n > 0$. Define $r'_1 = \ell'_2 = r'_2 = \cdots = r'_{n-1} = \ell'_n = s$ and if $m > 0$ then $\ell'_1 = \ell_1$ and $r'_n = r_m$ but if $m = 0$ then $\ell'_1 = r'_n = NIL$. Note that if $m = 0$, the edge from the first father (from the last father) of σ to σ does not have left-(right-)window. Then $(p_i, \ell'_i, r'_i) \in P_i$ $(1 \leqslant i \leqslant n)$. For $n = 0$, $\sigma(Q_1 \cdots Q_m) \to \sigma'$ is in $\hat{R}$ iff there exists $(q_i, \ell_i, \gamma_i) \in Q_i$, $1 \leqslant i \leqslant m$, such that $\gamma_i = \ell_{i+1}$, $1 \leqslant i < m$, and $\sigma(q_1 \cdots q_m) \to \sigma'$ is in R.

Before we formally prove the theorem, we introduce some pictorial convention of d-dags. A sequence of nodes $x_1, \dots, x_\ell$ is a *cut* of a d-dag d if x_i is directly to the left of x_{i+1} for every i, $1 \leqslant i < \ell$, and every path from the root to a leaf in d must contain exactly one node from this sequence. It follows that a sequence $x_1, \dots, x_\ell$ is a cut of d iff there is a partial d-dag d' such that d' is a subgraph of d and leaves$(d') = x_1 \cdots x_\ell$. Thus each cut "divides" d into d_1 and the subgraph of the rest d_2. We draw this division as

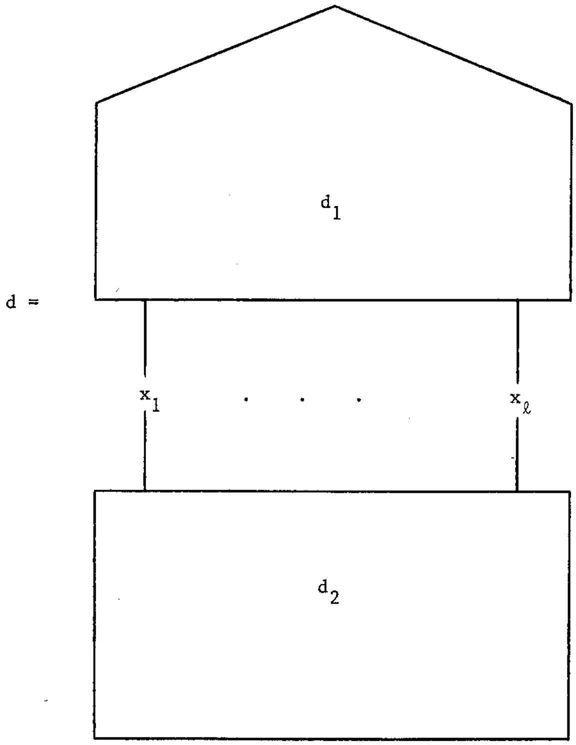
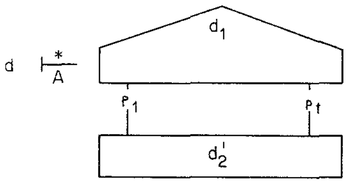


FIG. 14. A division of d-dag by a cut.

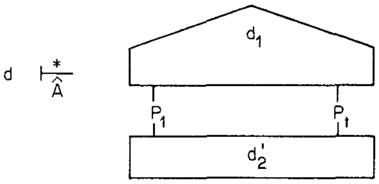
in Fig. 14. Notice that d_2 may not be a d-dag. It follows that if d in Fig. 14 appears in the computation sequence of a dag automaton $A = (Q, \Sigma, R) \in NB$ with $x_1, \ldots, x_\ell \in Q$, then every node in d_1 is labeled by a symbol of Σ whereas every node in d_2 is primed. Note that in this case all the leaves of d are in d_2.

Now the following lemma completes the proof of Theorem 3.6.

3.7. LEMMA. *Let $d \in D_\Sigma$, $p_1, \ldots, p_t \in Q$. Then*



if and only if there exist $P_1, \ldots, P_t \in \hat{Q}$ such that

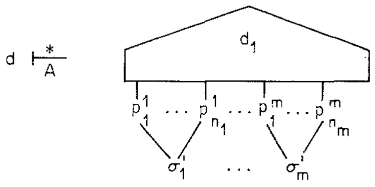


and there are $(p_i, \ell_i, \gamma_i) \in P_i$ $(1 \leqslant i \leqslant t)$ with $\gamma_i = \ell_{i+1}$ $(1 \leqslant i < t)$.

Proof. We first observe that if two distinct rules of dag automaton are applicable at some stage of the computation to an input, the order of their application is not significant at all. Hence, we may assume that given computation of A $(\hat{A})$ in this lemma is "normal" in the sense that A $(\hat{A})$ applies rules for all the leaves of d before it applies any other rules in the computation. Now both directions of the lemma are proved by the induction on the steps of the normal computation.

("Only if" part)

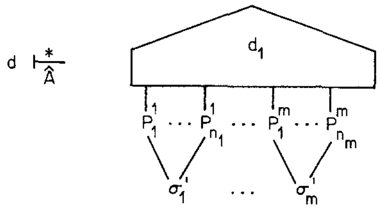
Basis. Assume that


d
$\vdash^*_A$
d_1
$p^1_1 \ldots p^1_{n_1} \ldots p^m_1 \ldots p^m_{n_m}$
$\sigma_1' \ldots \sigma_m'$

(3.1)

Then $s_i: \sigma_i \to [p_1^i \cdots p_{n_i}^i]\, \sigma_i' \in R$ $(1 \leqslant i \leqslant m)$. Hence $\sigma_i \to [P_1^i \cdots P_{n_i}^i]\, \sigma_i' \in \hat{R}$ with $(p_j^i, \ell_j^i, r_j^i) \in P_j^i$ $(1 \leqslant j \leqslant n_i)$, $\ell_1^i = r_{n_i}^i = NIL$ and $r_1^i = \ell_2^i = \cdots = r_{n_i-1}^i = \ell_{n_i}^i = s_i$ for each $1 \leqslant i \leqslant m$.

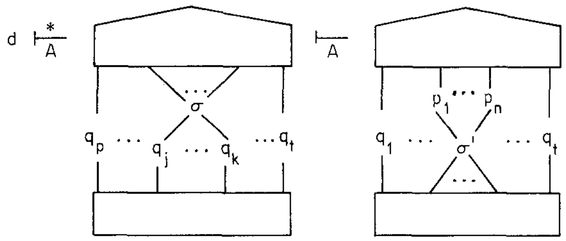
Therefore


d
$\overset{*}{\underset{\hat{A}}{\vdash}}$
d_1
$P_1^1 \cdots P_{n_1}^1 \cdots P_1^m \cdots P_{n_m}^m$
$\sigma_1' \cdots \sigma_m'$

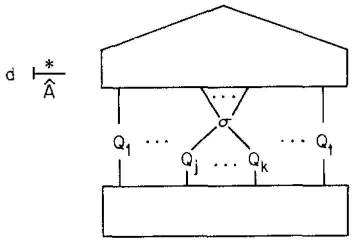
(3.2)

with $(p_j^i, \ell_j^i, r_j^i) \in P_j^i$ $(1 \leqslant j \leqslant n_i,\ 1 \leqslant i \leqslant m)$ as required.

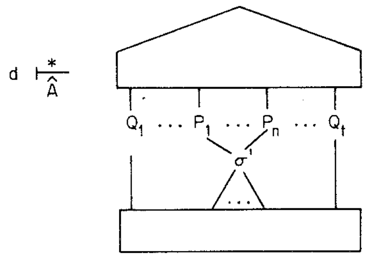
Induction Step. Assume that


d
$\overset{*}{\underset{A}{\vdash}}$
σ
$q_p \cdots q_j \cdots q_k \cdots q_t$
$\underset{A}{\vdash}$
$p_1 \cdots p_n$
$q_1 \cdots \sigma' \cdots q_t$

By the inductive hypothesis we may assume that


d
$\overset{*}{\underset{\hat{A}}{\vdash}}$
σ
$Q_1 \cdots Q_j \cdots Q_k \cdots Q_t$

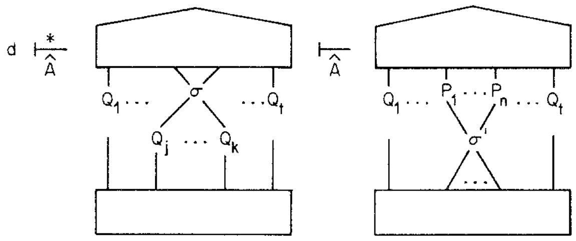
and there are $(q_i, \ell_i, r_i) \in Q_i$ $(1 \leqslant i \leqslant t)$ with $r_i = \ell_{i+1}$ $(1 \leqslant i < t)$. Obviously $s: \sigma(q_j \cdots q_k) \to [p_1 \cdots p_n]\, \sigma'$ is in R. By the definition of $\hat{R}$ there are $P_1, \ldots, P_n$ such that $\sigma(Q_j \cdots Q_k) \to [P_1 \cdots P_n]\, \sigma' \in R'$ and $(p_i, \ell_i', r_i') \in P_i$ with $\ell_1' = \ell_j = r_{j-1}'$, $r_n' = r_k = \ell_{k+1}$ and $r_i' = \ell_{i+1}'$ $(1 \leqslant i < n)$. Therefore,


d $\overset{*}{\underset{\hat{A}}{\vdash}}$
$Q_1 \cdots P_1 \cdots P_n \cdots Q_t$
σ'
$\cdots$

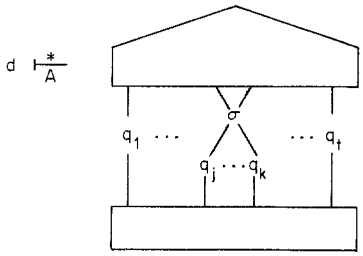
and the "only if" part holds.
("If" part)

Basis. From the definition of $\hat{R}$, it follows that if $\sigma \to [P_1 \cdots P_n]\,\sigma' \in \hat{R}$ and $(p_i, \ell_i, r_i) \in P_i$ $(1 \leqslant i \leqslant n)$ then $\ell_1 = r_n = NIL$ and $\ell_i = r_i$ $(1 < i < n)$. Therefore (3.2) obviously implies (3.1).

Induction Step. Assume that


d $\overset{*}{\underset{\hat{A}}{\vdash}}$
$Q_1 \cdots$
σ
$\cdots Q_t$
$Q_j \cdots Q_k$
$\underset{\hat{A}}{\vdash}$
$Q_1 \cdots P_1 \cdots P_n \cdots Q_t$
σ'
$\cdots$

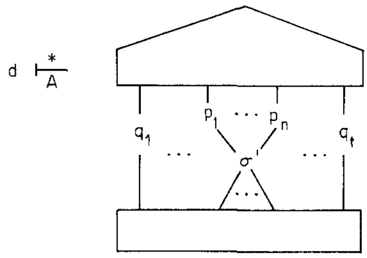
with $(q_i, \ell_i, r_i) \in Q_i$ $(1 \leqslant i < j$ or $k < j \leqslant t)$ and $(p_i, \ell'_i, r'_i) \in P_i$ $(1 \leqslant i \leqslant n)$ satisfying $r_i = \ell_{i+1}$ $(1 \leqslant i < j-1$ or $k < i < t)$, $r'_i = \ell'_{i+1}$ $(1 \leqslant i \leqslant n)$, $r_{j-1} = \ell'_1$ and $r'_n = \ell_{k+1}$. Then $\hat{R}$ must have the rule $\sigma(Q_j \cdots Q_k) \to [P_1 \cdots P_n]\,\sigma'$. By the definition of $\hat{R}$ there are $(q_i, \ell_i, r_i) \in Q_i$ $(j \leqslant i \leqslant k)$ such that $r_i = \ell_{i+1}$ $(j \leqslant i \leqslant k)$ and a rule s: $\sigma(q_j \cdots q_k) \to [p_1 \cdots p_n]\,\sigma' \in R$ such that $r'_1 = \ell'_2 = r'_2 = \cdots = \ell'_n = s$. Since $\ell_j = \ell'_1$ and $r_k = r'_n$, it follows that $r_i = \ell_{i+1}$ $(1 \leqslant i < t)$. Therefore, by the inductive hypothesis, we may assume that



$$\begin{tikzcd} \mathrm{RECOG}_D = \mathscr{L}(\mathrm{NT}) = \mathscr{L}(\mathrm{NB}) = \mathscr{L}(\mathrm{DB}) \arrow[d, dash] \\ \mathscr{L}(\mathrm{DT}) \end{tikzcd}$$

FIG. 15. Inclusion diagram of classes of *d*-dag languages defined by parallel dag automata.

Using the rule *s* we can derive:



This concludes the proof of Lemma 3.7 and Theorem 3.6. ∎

Languages in $\mathscr{L}(NB)$ $(=\mathscr{L}(NT)=\mathscr{L}(DB))$ are said to be *recognizable*, and the class will also be denoted by $RECOG_D$.

We summarize the results of this section in Fig. 15. It should be noted that a precisely similar diagram holds for the tree case. The solid lines denote proper containment.

4. FINITE STATE RELABELING

In this section, we modify the definition of a parallel dag automaton to define a simple transforming device, called finite state relabeling. Rather than priming the labels of the processed *d*-dag the automaton consistently relabels them by symbols of another doubly ranked alphabet. Finite state relabelings are an obvious extension of sequential machines (in the string case) and the tree version was used extensively by Engelfriet (1975) and Engelfriet *et al.* (1978). First we present the definition.

4.1 DEFINITION. A *finite state relabeling* (fsr) is a construct $T=(Q, \Sigma, \Delta, R)$, where Q is a finite set of states, Σ and Δ are disjoint doubly ranked alphabets and R is a finite set of rules. As in Definition 3.1, T is either deterministic or nondeterministic and either top-down or bottom-up depending on the form of rules in R. The rules of the top-down and the bottom-up fsr's are respectively of the form $[p_1 \cdots p_n]\, \sigma \rightarrow \delta(q_1 \cdots q_m)$ and

$\sigma(q_1 \cdots q_m) \to [p_1 \cdots p_n]\, \delta$ with $\sigma \in \Sigma_{nm}$ and $\delta \in \Delta_{nm}$. A *relabeling* is just a (total) deterministic single state relabeling. Alternatively, it can be defined as a function $h: \Sigma \to \Delta$ such that $\sigma \in \Sigma_{nm}$ implies $h(\sigma) \in \Delta_{nm}$ for every $\sigma \in \Sigma$. ▮

For a finite state relabeling T, the concept of configuration and the relations $\vdash_T$ and $\vdash_T^*$ are defined similarly to those of finite dag automata, see Definition 3.1.

Let T be a finite state relabeling from Σ to Δ; then for $L \subseteq D_\Sigma$, $T(L) = \{g \in D_\Delta \mid d \vdash_T^* g$ for some $d \in L\}$ and for $L \subseteq D_\Delta$, $T^{-1}(L) = \{d \in D_\Sigma \mid d \vdash_T^* g$ for some $g \in L\}$. We identify the finite state relabeling T with the transformation $\{(d, g) \mid d \in D_\Sigma, g \in D_\Delta$ and $d \vdash_A^* g\}$. The domain and the range of T, denoted respectively as dom(T) and ran(T), are $T^{-1}(D_\Delta)$ and $T(D_\Sigma)$.

TQREL (*DTQREL*) will denote the set of all transformations definable by nondeterministic (deterministic) top-down finite state relabelings; *BQREL* and *DBQREL* are defined similarly in the bottom-up case. An argument similar to that of Theorem 3.4 proves the following fact.

4.2. Lemma. *TQREL* $=$ *BQREL*. ▮

Henceforth, both *TQREL* and *BQREL* will be denoted by *QREL*.

We now prove the recognizability of the domains of finite state relabelings.

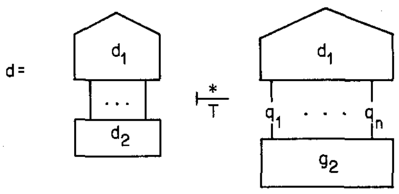
4.3. Lemma. *Let T be an element of QREL, DTQREL or DBQREL; then* dom(T) *is recognized by an automaton of corresponding type.*

Proof. Let $T = (Q, \Sigma, \Delta, R) \in BQREL$. We define $A = (Q, \Sigma, R') \in NB$ as follows. R' has the rule $\sigma(q_1 \cdots q_m) \to [p_1 \cdots p_n]\, \sigma'$ if and only if R has the rule $\sigma(q_1 \cdots q_m) \to [p_1 \cdots p_n]\, \delta$ for some $\delta \in \Delta$, where $q_1, \ldots, q_m$, $p_1, \ldots, p_n \in Q$. Obviously, $L(A) = \text{dom}(T)$ and A is deterministic if T is. The top-down case is similar. ▮

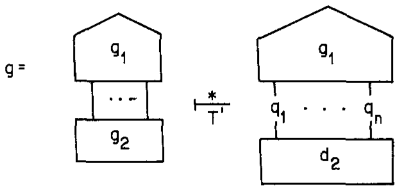
In the next lemma, we show that the inverse of a finite state relabeling is also a finite state relabeling.

4.4 Lemma. *For each $T \in QREL$ from Σ to Δ, there is $T' \in QREL$ from Δ to Σ such that for $d \in D_\Sigma$ and $g \in D_\Delta$, $d \vdash_T^* g$ if and only if $g \vdash_{T'}^* d$.*

Proof. Let $T = (Q, \Sigma, \Delta, R)$ in *BQREL*. Define $T' = (Q, \Delta, \Sigma, R')$ in *BQREL*, where R' has a rule $\delta(q_1 \cdots q_m) \to [p_1 \cdots p_n]\, \sigma$ if and only if R has a rule $\sigma(q_1 \cdots q_m) \to [p_1 \cdots p_n]\, \delta$ with $\sigma \in \Sigma$, $\delta \in \Delta$ and $q_1, \ldots, q_m$, $p_1, \ldots, p_n \in Q$. To prove the lemma formally, one can use an inductive argument on the steps of relabeling process to prove the equivalence:



if and only if



where $d_1(g_1)$ is a partial d-dag of $d(g)$ and $d_2(g_2)$ is the subgraph of the rest of $d(g)$. ▮

Remark. It follows from the lemma that for $L \subseteq D_\Sigma$, $T(L) = (T')^{-1}(L)$ and for $K \subseteq D$, $T^{-1}(K) = T'(K)$. In particular, $\mathrm{dom}(T) = \mathrm{ran}(T')$ and $\mathrm{ran}(T) = \mathrm{dom}(T')$. ▮

The previous two lemmas together with the Remark above give the next corollary.

4.5. COROLLARY. *For each $T \in QREL$, $\mathrm{ran}(T)$ is recognizable.* ▮

4.6. LEMMA. *Finite state relabelings of each class are closed under composition.* ▮

Proof. Let $T_1 = (Q_1, \Sigma, \Delta, R_1)$ and $T_2 = (Q_2, \Delta, \Gamma, R_2)$ be finite state relabelings in *BQREL*, and let us assume that Σ and Γ are disjoint. The finite state relabeling T to realize the composition of T_1 and T_2, denoted as $T_1 \circ T_2$ (first T_1 and then T_2), is defined using the standard Cartesian product construction as follows: $T = (Q_1 \times Q_2, \Sigma, \Gamma, R)$, where R has a rule

$$\sigma((q_1^1, q_1^2) \cdots (q_m^1, q_m^2)) \to [(p_1^1, p_1^2) \cdots (p_n^1, p_n^2)]\,\tau$$

if and only if R_1 has a rule $\sigma(q_1^1 \cdots q_m^1) \to [p_1^1 \cdots p_n^1]\,\delta$ and R_2 has a rule $\delta(q_1^2 \cdots q_m^2) \to [p_1^2 \cdots p_n^2]\,\tau$, where $\sigma \in \Sigma$, $\tau \in \Gamma$, $\delta \in \Delta$, $q_1^i, \ldots, q_m^i, p_1^i, \ldots, p_n^i \in Q_i$, $i = 1, 2$. The proof of $T = T_1 \circ T_2$ is straightforward. Note that if T_1 and T_2 are deterministic, T is deterministic as well. Similar construction works for the top-down finite state relabelings. ▮

5. Two-Way dag-Walking Automata

In this section, we investigate dag automata of the usual sequential type, which are called two-way dag-walking automata. The two-way dag automata have a single pointer into the input d-dag. They are allowed to walk freely on the d-dag except that when moving up they must go through the edge by which they came down. Note that this is no restriction in the case of trees. This requirement is especially meaningful when the automaton is allowed to temporarily write some information on the d-dag (compare to remarks at the end of Aho and Ullman (1971) and also to Engelfriet *et al.* (1978)). Thus, the automata which are allowed to use the input as temporary storage are restricted to have a synchronized push-down store between the root of the d-dag and the current location of the pointer.

5.1. Definition. A *two-way push-down dag-walking automaton* is a construct $A=(Q,\Sigma,\Gamma,\delta,q_0,\gamma_0,F)$, where Q is a finite set of states, Σ is a doubly ranked alphabet, Γ is the pushdown alphabet, $q_0\in Q$ is the initial state of A, $\gamma_0\in\Gamma$ is the initial push-down symbol and $F\subseteq Q$ is the subset of final states. δ is a mapping from $Q\times\Sigma\times\Gamma$ to finite subsets of $Q\times D$ where

$$D=\{-i\,|\,i\geqslant 1\}\cup\{(i,\gamma_1\gamma_2)\,|\,\gamma_1,\gamma_2\in\Gamma, i\geqslant 1\}. \quad \blacksquare$$

The set D specifies the direction of the move for the next step indicating the ith father and the ith son by $-i$ and i, respectively, for each $i\geqslant 1$.

A *configuration* of a two-way push-down dag-walking automaton $A=(Q,\Sigma,\Gamma,\delta,q_0,\gamma_0,F)$ on a d-dag $d\in D_\Sigma$ is a quadruple $\langle q,d,\alpha,\beta\rangle$, where $q\in Q$, α is a path in d, $\alpha=(n_1,n_2,\ldots,n_k)$ with n_1 the root of d and n_k the node currently scanned, and $\beta=\gamma_1\cdots\gamma_k\in\Gamma^*$ is the contents of the push-down store. Let $C_1=\langle q,d,(n_1,\ldots,n_k),\beta_1\gamma\rangle$ and $C_2=\langle p,d,(n_1,\ldots,n_\ell),\beta_2\rangle$ be two configurations with n_k labeled by σ and $\gamma\in\Gamma$. Then $C_1\vdash_A C_2$ if either of the following holds.

(1) $(p,(j,\gamma_1'\gamma_2'))\in\delta(q,\sigma,\gamma)$, $\beta_2=\beta_1\gamma_1'\gamma_2'$, $\ell=k+1$ and n_ℓ is the jth son of n_k.

(2) $(p,-j)\in\delta(q,\sigma,\gamma)$, $\beta_2=\beta_1$, and if $k>1$, $\ell=k-1$ and n_ℓ is the jth father of n_k. If $k=1, j=-1$; thus $\langle q,d,(n_1),\gamma\rangle\vdash_A\langle p,d,(\,),\lambda\rangle$.

Even though the automaton knowns at each stage to which father it goes, $\delta(q,\sigma,\gamma)$ can contain more than one pair of the type $(p,-j)$ with distinct j, for $p,q\in Q$, $\sigma\in\Sigma$, $\gamma\in\Gamma$. It can move up to the jth father of a current node by selecting a pair $(p,-j)$ if the last time it came down to the node was from its jth father. A is *deterministic* if the following conditions are satisfied:

(i) $(p,(j,\gamma_1\gamma_2))\in\delta(q,\sigma,\gamma')$ implies $\delta(q,\sigma,\gamma')=\{(p,(j,\gamma_1\gamma_2))\}$.

(ii) $(p_1, -j_1)$, $(p_2, -j_2) \in \delta(q, \sigma, \gamma)$ implies $j_1 \neq j_2$. Note that in general a deterministic δ is not a partial function form $Q \times \Sigma \times \Gamma$ to $Q \times D$ as in the tree case (see Engelfriet *et al.* (1978)). This is because such a restriction would force the automaton to move up in the same direction and the same state each time it comes to some node in the same state (with the same push-down symbol) and is about to move up. This would make our automata unduly awkward. On the other hand note that the above restrictions force a deterministic behavior for any input d-dag. The relation $\vdash_A^*$ is the reflexive–transitive closure of $\vdash_A$ and the d-dag language recognized by A is $L(A) = \{d \in D_\Sigma \mid (q_0, d, (n_1), \gamma_0) \vdash_A^* (p, d, (\,), \lambda)$ with some $p \in F\}$; where n_1 is the root of d.

A two-way push-down dag-walking automaton is a *finite state dag-walking automaton* if its push-down alphabet contains a single symbol. In this case the push-down is redundant and the configurations will be $\langle q, d, \alpha \rangle$ with $q \in Q$, $d \in D_\Sigma$ and α a path from the root of d to some node.

Again as in the case of parallel dag automata, definitions of configurations and acceptance q input by dag-walking automlata are not actually restricted to d-dags, they are valid for DOAGs.

$2N$–PD and $2D$–PD denote respectively the sets of all nondeterministic and deterministic two-way push-down automata; similarly, $2N$ and $2D$ are the versions without the push-down facility.

5.2. Example. The following deterministic two-way push-down automaton $A = (Q, \Sigma, \Gamma, \delta, q_a, A_0, F)$ recognizes the language L of Example 3.2. $Q = \{q_a, q_b, p\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{A_0, A_1, A_2\}$ and $F = \{p\}$. The transition function δ is as follows.

For $a \in \Sigma_{02} \cup \Sigma_{12}$,

$$\delta(q_a, a, A_0) = \{q_a, (1, A_1 A_0))\},$$
$$\delta(p, a, A_1) = \{(q_a, (2, A_2 A_0))\},$$
$$\delta(p, a, A_2) = \{(p, -1)\}.$$

For $b \in \Sigma_{11}$,

$$\delta(q_a, b, A_0) = \{(q_b, (1, A_1 A_0))\},$$
$$\delta(p, b, A_1) = \{(q_b, (2, A_2 A_0))\},$$
$$\delta(p, b, A_2) = \{(p, -1)\}.$$

For $c \in \Sigma_{20}$,

$$\delta(q_a, c, A_0) = \delta(q_b, c, A_0) = \{(p, -1), (p, -2)\}.$$

When the automaton arrives at a node labeled by a from one of its fathers, it then moves down to its first son; after coming back to the node from the first son, it visits the second son next; moving up from the second son, it finally goes up to the father from which it came down. The pushdown is used (at a) to remember how many sons have been visited: A_i is for i sons, $i = 1, 2$. The computation of the automaton on a d-dag is illistrated in Fig. 16. The path from the root of the d-dag to the currently visited node is indicated by a bold line in the figure. ▮

5.3. Example. Let $\Sigma_{02} = \Sigma_{12} = \{A\}$ and $\Sigma_{10} = \{a, b\}$. Consider the following d-dag (tree) language L over Σ: $L = \{t \in D_\Sigma \mid \text{leaves}(t) \in a^*\}$. The language L is recognized by the following two-way push-down dag (tree) walking automaton $B = (Q, \Sigma, \Gamma, \delta, q_0, B_1, F)$. $Q = \{q_0, q_1, q_2\}$, $\Gamma = \{B_1, B_2\}$ and $F = \{q_1\}$. The transition function δ is defined as follows.

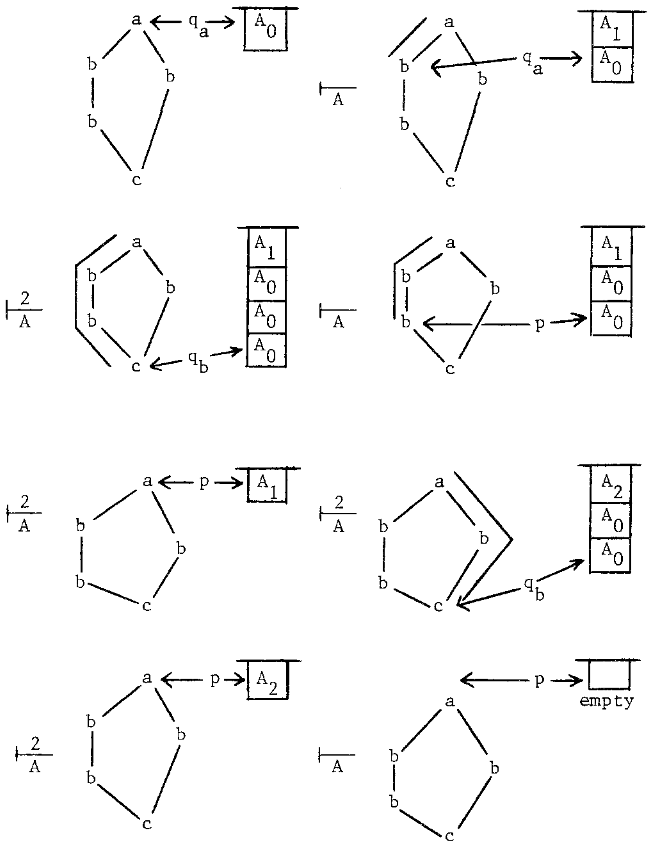


Fig.16. An example of dag-walking automaton.

For A,

$$\delta(q_0, A, X) = \{(q_0, (1, XB_1))\},$$
$$\delta(q_1, A, X) = \{(q_0, (2, XB_2))\},$$
$$\delta(q_2, A, B_1) = \{(q_1, -1)\},$$
$$\delta(q_2, A, B_2) = \{(q_2, -1)\}.$$

For a,

$$\delta(q_0, a, B_0) = \{(q_1, -1)\},$$
$$\delta(q_0, a, B_1) = \{(q_2, -1)\},$$

where $X \in \{B_1, B_2\}$.

B traverses an input tree in pre-order. B is blocked if it reaches a leaf labeled by b; otherwise, it visits all the node of the input and accepts it. B arrives at an internal node A in q_0 for the first time; it is in $q_1(q_2)$ at the node when coming up from its first (second) son. B_1 and B_2 are used in the pushdown to remember which son of the node is currently visited. ∎

We shall now compare the relative recognition power of the various two-way dag walking automata. The first result states that in the presence of push-down facility, nondeterminism has no more power than determinism. To prove this, we extend the method of transition tables (Hopcroft and Ullman, 1967; Shepherdson, 1959) to d-dags.

5.4. THEOREM. $\mathscr{L}(2N - PD) = \mathscr{L}(2D - PD)$.

Proof. We have to show $\mathscr{L}(2N - PD) \subseteq \mathscr{L}(2D - PD)$. Given nondeterministic push-down dag-walking automaton $A = (Q, \Sigma, \Gamma, \delta, q_0, \gamma_I, F)$, we construct the following equivalent deterministic automaton $\hat{A}$. The automaton $\hat{A}$ simulates A by computing the transition tables of A at each node of the input. $\hat{A}$, after arriving at a node from its jth father, visits all its sons $\tau_1, ..., \tau_m$ and all their descendants from left to right, see Fig. 17. When moving back from τ_i to σ, $\hat{A}$ computes the transition table M_i: $Q \times \Gamma \to \mathscr{P}(Q)$, with the following meaning: when A goes down from σ to τ_i in a state q and pushes γ at the top of the stack there is a computation of A which takes A back to σ in state p, for each $p \in M_i(q, \gamma)$; if there is no such a computation $M_i(q, \gamma) = \varnothing$. After computing all the functions $M_1, ..., M_m$, $\hat{A}$ computes the transition table M_j: $Q \times \Gamma \to \mathscr{P}(Q)$ of the node σ for its jth father as follows: $p \in M^j(q, \gamma)$ if and only if there are $q_1, ..., q_k$, $p_1, ..., p_k$, $\gamma_1, ..., \gamma_k$, $\gamma'_1, ..., \gamma'_k$, $k \geqslant 0$ such that $(p_\ell, (i_\ell, \gamma_\ell \gamma'_\ell)) \in \delta(q_{\ell-1}, \sigma, \gamma_{\ell-1})$, $q_\ell \in M_{i_\ell}(p_\ell, \gamma'_\ell)$ for $1 \leqslant \ell \leqslant k$ with $q_0 = q$ and $\gamma_0 = \gamma$ and $(p, -j) \in \delta(q_k, \sigma, \gamma_k)$.

Suppose A visits σ from its jth father in state q and γ at the top of the

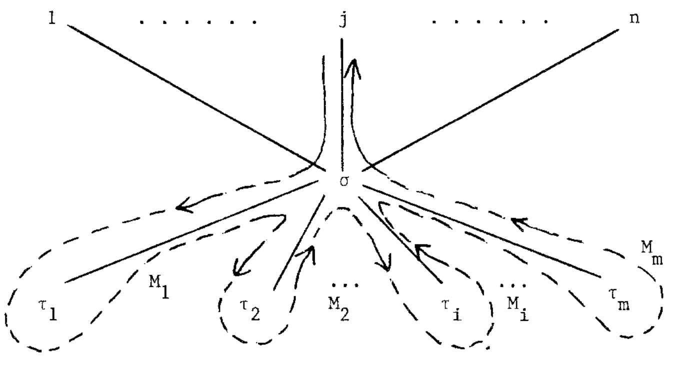


FIG. 17. The computation of $\hat{A}$.

stack, and then (for the first time) it moves up to the father in state p. Between these transitions, A visits the sons of σ, $\tau_{i_1}, \ldots, \tau_{i_k}$ in this order. In the case of $k = 0$, A immediately goes up to the father by $(p, -j) \in \delta(q, \sigma, \gamma)$ (in particular when $m = 0$). Thus, assuming that $M_1, \ldots, M_m$ are the correct transition tables for the sons of σ and σ, M^j is obviously the correct transition table of σ and its jth father. The acceptance of an input d-dag is determined by the function M^1 for the root of the d-dag by checking $M^1(q_0, \gamma_I) \cap F \neq \varnothing$, where γ_I is the initial push-down symbol of A.

The formal definition of $\hat{A}$ follows. Let $\mathscr{F}$ be the set of functions from $Q \times \Gamma$ to $\mathscr{P}(Q)$; $\mathscr{F}^i = \{\langle f_1 \cdots f_i \rangle \mid f_j \in \mathscr{F}, 1 \leqslant j \leqslant i\}$. Then $\hat{A} = (\{q_0\} \cup \mathscr{F}, \Sigma, \mathscr{G}, \delta, q_0, \langle\,\rangle, \hat{F})$ where $\mathscr{G} = \bigcup_{i=0}^{N} \mathscr{F}^i$ with N being the maximal tail-rank of symbols in Σ and $\langle\,\rangle$ is the unique element of $\mathscr{F}^0$. For the sake of convenience, a root is assumed here to have the head-rank 1. The transition function $\hat{\delta}$ is defined as follows.

For $\sigma \in \Sigma_{nm}$ $(m > 0)$

$$\hat{\delta}(q_0, \sigma, \langle\,\rangle) = \{(q_0, (1, \langle\,\rangle \langle\,\rangle))\},$$

$$\hat{\delta}(M, \sigma, \langle M_1 \cdots M_{i-1} \rangle) = \{(q_0, (i+1, \langle M_1 \cdots M_{i-1} M \rangle \langle\,\rangle))\}$$

$$\text{for } 1 \leqslant i < m \text{ and } M, M_1, \ldots, M_{i-1} \in \mathscr{F},$$

$$\hat{\delta}(M_m, \sigma, \langle M_1 \cdots M_{m-1} \rangle) = \{(M^j, -j) \mid 1 \leqslant j \leqslant n\}.$$

For $\sigma \in \Sigma_{n0}$

$$\hat{\delta}(q_0, \sigma, \langle\,\rangle) = \{(M^j, -j) \mid 1 \leqslant j \leqslant n\}.$$

Finally, $\hat{F} = \{M \in \mathscr{F} \mid M(q_0, \gamma_I) \cap F \neq \varnothing$, where γ_I is an initial push-down symbol of $A\}$. ∎

We next show that the push-down facility increases the recognition power of two-way walking automata. This is proved by showing that no finite state dag-walking automaton can recognize the language L of Example 5.3.

5.5. THEOREM. $\mathscr{L}(2N) \subsetneq \mathscr{L}(2D-PD)$.

Proof. Suppose a nondeterministic finite state dag-walking automaton $B=(Q, \Sigma, \gamma, q_0, F)$ recognizes the language L of Example 5.3. Let k be the cardinality of Q and let $d \in D_\Sigma$ be a balanced binary tree of height $> \log_2(k-1)$, see Fig. 18. Then $|\text{leaves}(d)| > k$. Obviously, $d \in L$. Therefore there is an accepting computation of B on d, and in every accepting computation B must visit all the leaves. Otherwise B would also accept the tree obtained from d by changing the label of an unvisited leaf from a to b. Since $|\text{leaves}(d)| > k$, B visits two different leaves in the same state during the computation. Let these nodes be n_a and n_b; and let n_c be their lowest common ancestor. Let t_1 and t_2 be the subtrees of d at the node n_c, see Fig. 19. Assuming without loss of generality that B visits n_a before n_b, the accepting computation of B on d is then of the form:

$$\begin{aligned}
(q_0, d, (n_1)) &\overset{*}{\underset{B}{\vdash}} C_1 = (q, d, (n_1, \dots, n_c, \dots, n_a)),\\
&\overset{*}{\underset{B}{\vdash}} C_2 = (p_1, d, (n_1, \dots, n_c)),\\
&\overset{*}{\underset{B}{\vdash}} C_3 = (q, d, (n_1, \dots, n_c, \dots, n_b)),\\
&\overset{*}{\underset{B}{\vdash}} C_4 = (p_2, d, (n_1, \dots, n_c)),\\
&\overset{*}{\underset{B}{\vdash}} (q_f, d, ()),
\end{aligned}$$

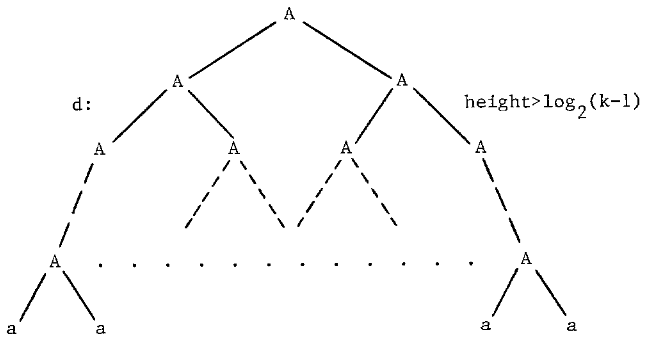


FIG. 18. A balanced binary tree in L.

where q, p_1, p_2, $q_f \in Q$, $q_f \in F$ and C_4 is the first configuration after C_3 in which A moves outside of t_2. We show now that there is another computation which skips the part $C_1 \vdash_B^* C_2 \vdash_B^* C_3$ and directly goes from C_1 to C_4. For each configuration $C = (p, d, (n_1, ..., n_c, n_{c+1}, ..., n_e))$ such that $C_3 \vdash_B^* C \vdash_B^* C_4$, we define a configuration $C' = (p, d, (n_1, ..., n_c, n'_{c+1}, ..., n'_e))$ inductively as follows: (i) $C'_3 = C_1$. (ii) Suppose we have defined C' for C; if $C \vdash_B \hat{C} = (\hat{p}, d,, (n_1, ..., n_c, n_{c+1}, ..., n_e, n_{e+1}))$, where n_{e+1} is the left (right) son of n_e, then $\hat{C}' = (\hat{p}, d, (n_1, ..., n_c, n'_{c+1}, ..., n'_e, n'_{e+1}))$, where n'_{e+1} is the left (right) son of n'_e; if $C \vdash_B \hat{C} = (\hat{p}, d, (n_1, ..., n_c, n_{c+1}, ..., n_{e-1}))$, then $\hat{C}' = (\hat{p}, d, (n_1, ..., n_c, n'_{c+1}, ..., n'_{e-1}))$. It follows that for each configuration C such that $C_3 \vdash_B^* C \vdash_B^* C_4$, $C_1 \vdash_B^* C' \vdash_B^* C_4$. Thus we have found another successful computation in which one visit of the node n_b is skipped. As long as a computation of B visits more than k leaves, we can always find a new computation of B in the way outlined above. However, since every successful computation must visit all the leaves, this leads to a contradiction. ▮

In the case of finite state dag-walking automata, without the push-down facility, nondeterminism gives us more power than determinism. Let L be the language of Example 5.3 and K be $D_\Sigma - L$, where Σ is the alphabet over which L is defined. Thus $K = \{d \in D_\Sigma \mid \text{leaves}(d)$ has at least one occurrence of $b\}$. In the next theorem, we show that K can be recognized by a nondeterministic automaton in $2N$, but by no automaton in $2D$.

5.6. THEOREM. $\mathscr{L}(2D) \subsetneq \mathscr{L}(2N)$.

Proof. The language K is recognized by the following automaton in $2N$.

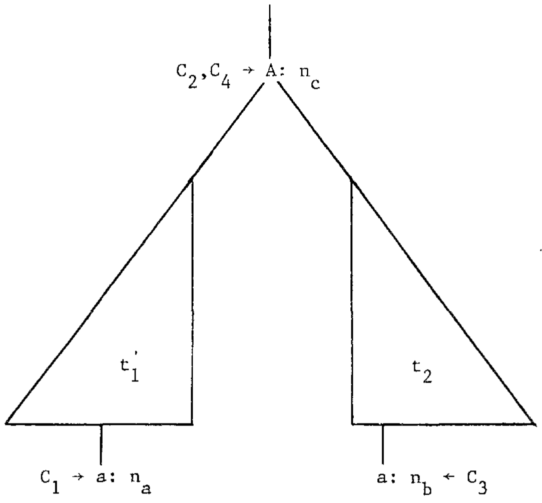

$C_2, C_4 \rightarrow A$: n_c
t'_1
t_2
$C_1 \rightarrow a$: n_a
a: $n_b \leftarrow C_3$

FIG. 19. Computation of B.

It first nondeterministically visits one leaf of the input d-dag and then comes back to the root with the label of the leaf. The automaton accepts the input d-dag if the label was "b."

Suppose K is recognized by the automaton $B = (Q, \Sigma, \delta, q_0, F)$ in $2D$. As before, let k be the cardinality of Q and consider the balanced binary tree $d \in D_\Sigma$ of height h, see Fig. 20. For $d \notin K$, B must reject d by either ending its computation in a nonaccepting state or by entering an infinite loop or ends at a node other than the root. In any event, B must visit all the leaves of such a d-dag, since otherwise we can change the label of the unvisited node to b, as in the proof of Theorem 5.5. Let $a_1, a_2,...$ be the sequence of leaves of d visited by B in his order; some leaves may be repeated in the sequence.

For any two leaves a_i and a_j, the *distance between a_i and a_j* is defined to be the length of the path from a_i (or a_j) to their nearest common ancestor. If $a_i = a_j$, then the distance between them is 0. Note that for each leaf a_i, the number of leaves of d (including a_i) whose distance from a_i is less than or equal to ℓ is precisely 2^ℓ. Therefore, if h is large enough, there is an i such that the distance between a_i and a_{i+1} is greater than k. Let x be the nearest common ancestor of a_i and a_{i+1} (in d). In the transition process from a_i to x, B visits at least $k + 1$ different internal nodes, all of which are labeled by A. Therefore, B must visit two different internal nodes, n_1 and n_2 say, n_2 being above n_1, in the same state. Since B does not visit any leaves between a_i and x, it must repeat the computation between n_1 and n_2 until it arrives at the root of d. Thus B goes from a_i to a_{i+1} through the root of d. Hence between two visits to the root at most 2^k different leaves are visited; because of determinism these leaves are determined completely by the state in which B is at the first of these two visits. This means that B can visit at most $k \cdot 2^k$ different leaves. This contradiction proves the theorem. ▌

The inclusion relationships between the two-way walking autmata are summarized in Fig. 21. The diagram holds unmodified for the tree case (in fact we used tree languages for Theorems 5.5 and 5.6).

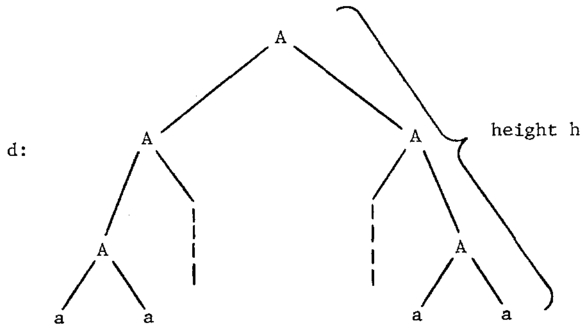


FIG. 20. A balanced binary tree not in K.

$$\begin{tikzcd}
\mathcal{L}(\text{2N-PD}) = \mathcal{L}(\text{2D-PD}) \arrow[d, dash] \\
\mathcal{L}(\text{2N}) \arrow[d, dash] \\
\mathcal{L}(\text{2D})
\end{tikzcd}$$

FIG. 21. The hierarchy of two-way dag-walking automata.

6. COMPARISON

In this section we compare the recognition power of parallel dag automata and two-way dag-walking automata. It turns out that the synchronized push-down store is precisely what is needed to handle the parallelism in DB. On the other hand, the parallelism in DT cannot be handled by $2N$ and the two-way motion capability cannot be handled by DT.

First, we show that a recognizable language can be recognized by an automaton in $2N-PD$.

6.1. THEOREM. $\mathscr{L}(DB) \subseteq \mathscr{L}(2N-PD)$.

Proof. Let $L \in \mathscr{L}(DB)$ be a language over Σ and let $A = (Q, \Sigma, R)$ be a parallel dag automaton in DB which recognizes L. We construct a two-way push-down dag-walking automaton $\hat{A}$ to recognizes L. Suppose the rule $\sigma(q_0 \cdots q_m) \rightarrow [p_1 \cdots p_n]\, \sigma'$ is applied by A, see Fig. 8. Now, to simulate A, $\hat{A}$ when coming down to the node σ from its jth father τ_j, visits $\sigma_1,\ldots, \sigma_m$ in this order, computing the states $q_1,\ldots, q_m$. When computing q_{i+1}, the already computed states $q_1,\ldots, q_i$ are stored in the push-down at σ until q_m is found. Then $\hat{A}$ is able to compute p_j and move up to τ_j in that state. A simple inductive argument on the steps of computation of A and $\hat{A}$ shows that (on a given input) $\hat{A}$ moves up to τ_j in state p_j, $1 \leqslant j \leqslant n$, if and only if A in its computation arrives in state p_j at τ_j (the jth father of σ).

The formal definition of $\hat{A} = (\hat{Q}, \Sigma, \hat{\Gamma}, \delta, q_0, \langle\,\rangle, \hat{F})$ is as follows: $\hat{Q} = Q \cup \{q_0, q_f\}$, where q_0, $q_f \notin Q$, $\Gamma = \bigcup_{i=0}^{N} \{\langle q_1 \cdots q_i\rangle \mid q_j \in Q, 1 \leqslant j \leqslant i\}$, N is the maximal tail-rank of symbols of Σ, $\langle\,\rangle \in \Gamma$ is the initial symbol and $\hat{F} = \{q_f\}$. The transition functions are defined as follows:

For $\sigma \in \Sigma_{nm}$ $(m > 0)$,

$$\begin{aligned}
\delta(q_p, \sigma, \langle\,\rangle) &= \{(q_0, (1, \langle\,\rangle\langle\,\rangle))\},\\
\delta(p, \sigma, \langle q_1 \cdots q_{i-1}\rangle) &= \{(q_0, (i+1, \langle q_1 \cdots q_{i-1} p\rangle\langle\,\rangle))\}\\
&\quad \text{for } 1 \leqslant i \leqslant m,\ p, q_1, \ldots, q_{i-1} \in Q,\\
\delta(q_m, \sigma, \langle q_1 \cdots q_{m-1}\rangle) &= \{(p_i, -i) \mid 1 \leqslant i \leqslant n,\\
&\quad \text{and } \sigma(q_1 \cdots q_m) \to [p_1 \cdots p_n]\,\sigma' \in R\}\\
&\quad \text{for } n > 0,\\
\delta(q_m, \sigma, \langle q_1 \cdots q_{m-1}\rangle) &= \{(q_f, -1)\}\\
&\quad \text{for } n = 0 \text{ if } \sigma(q_1 \cdots q_m) \to \sigma' \in R.
\end{aligned}$$

For $\sigma \in \Sigma_{n0}$,

$$\delta(q, \sigma, \langle\,\rangle) = \{(p_i, -i) \mid 1 \leqslant i \leqslant n \text{ and } \sigma \to [p_1 \cdots p_n]\,\sigma' \in R\}. \quad \blacksquare$$

In the proof of Theorem 5.4 we constructed a deterministic push-down dag walking automaton $\hat{A}$ which is equivalent to a given $A \in 2N-PD$. In the next theorem we construct a deterministic bottom-up dag automaton A' equivalent to this $\hat{A}$.

6.2. Theorem. $\mathscr{L}(2N-PD) \subseteq \mathscr{L}(DB)$.

Proof. Let us first recall some relevant properties of the automaton $\hat{A} = (\{q_0\} \cup \mathscr{F}, \Sigma, \mathscr{G}, \hat{\delta}, q_0, \langle\,\rangle, F)$ of Theorem 5.4. The state of $\hat{A}$ is always q_0 when it is moving down in the input d-dag; when it arrives at the node σ from its jth father $(1 \leqslant j \leqslant n)$, $\hat{A}$ visits its sons from left to right computing the functions $M_1, \ldots, M_m$ and then moves back to the jth father with M^j, where $M_1, \ldots, M_m, M^j \in \mathscr{F}$, see Fig. 17. The bottom-up parallel dag automaton A' to be constructed simulates such a computation by the rule $\sigma(M_1 \cdots M_m) \to [M^1 \cdots M^n]\,\sigma'$. The formal definition of A' is as follows: $A' = (\mathscr{F}, \Sigma, R)$, where for $\sigma \in \Sigma_{nm}$ $(n > 0)$, $\sigma(M_1 \cdots M_m) \to [M^1 \cdots M^n]\,\sigma' \in R$ if $\hat{\delta}(M_m, \sigma, M_1 \cdots M_{m-1}) = \{M^i, -i) \mid 1 \leqslant i \leqslant n\}$, and for $\sigma \in \Sigma_{0m}, \sigma(M_1 \cdots M_m) \to \sigma' \in R$ if $\hat{\delta}(M_m, \sigma, M_1 \cdots M_{m-1}) = \{(M^1, -1)\}$ with $M^1 \in \hat{F}$. Note that for $\sigma \in \Sigma_{nm}$ and, $M_1, \ldots, M_m$ in $\mathscr{F}$, the states of the right-hand side $M^1, \ldots, M^n$ are uniquely determined (see the proof of Theorem 5.4). Thus, A' is in DB. By an inductive argument on the steps of computation one can show that for each node $\sigma \in \Sigma_{*m}$ of an input dag, $\hat{A}$ computes the transition tables $M_1, \ldots, M_m$ (by moving m times down to the sons of σ) if and only if A' (operating on the same input d-dag) applies at σ a rule with left-hand side $\sigma(M_1 \cdots M_m)$. From the rules of A' for the root, it follows that $L(A') = L(\hat{A})$. $\blacksquare$

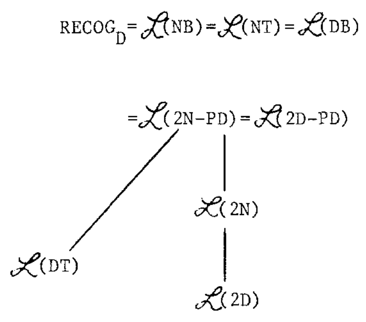


FIG. 22. The inclusion diagram for various classes of d-dag laguages.

We have now established that $RECOG_D = \mathscr{L}(2N-PD)$. The classes $\mathscr{L}(DT)$ and $\mathscr{L}(2N)$ or $\mathscr{L}(2D)$ are now shown to be incomparable. First, it is easy to see that the language L in the proof of Theorem 3.5 is recognized by an automaton in $2D$. On the other hand, the following automaton in DT can recognize the language L of Example 5.3. $A = (Q, \Sigma, R)$, where $Q = \{q\}$ and R is $\{A \to A'(qq), [q]\, A \to A'(qq), [q]\, a \to a'\}$.

In Fig. 22 we bring together the classes of d-dag languages definable by the various dag automata into an inclusion diagram. Observe that the proofs of Theorems 6.1 and 6.2 work for trees as a restricted case; thus this diagram also holds for tree.

The next theorem characterizes the difference between $RECOG_D$ on the one hand and $\mathscr{L}(DT)$, $\mathscr{L}(2N)$ and $\mathscr{L}(2D)$ on the other. Intuitively, it says that there is no difference between these classes of d-dag languages as far as the underlying graph structure is concerned. It is the way in which the d-dags are labeled that makes one d-dag language more difficult to recognize than the other.

6.3. THEOREM. *Let K be any of DT, $2N$ or $2D$. For every $L \in RECOG_D$ there exist a language $L' \in \mathscr{L}(K)$ and a relabeling h such that $L = h(L')$. Moreover, there is a finite state relabeling T such that $L' = T(L)$.*

Proof. Let $A = (Q, \Sigma, R)$ be a nondeterministic top-down dag automaton recognizing L. From A, we construct a nondeterministic top-down finite state relabeling T such that $L' = T(L)$. T relabels a node $\sigma \in \Sigma_{n^*}$ of an input d-dag in L by $\langle \sigma, \gamma, i \rangle$, where γ is any rule of A applicable at σ and i is an integer indicating that σ is the ith son of its father(s). Note that by the definition of d-dags if σ has more than one father, it is the only son of each father; thus $i = 1$. The formal definition of T is as follows: $T = (Q \times \{1,..., M\}, \Sigma, \Sigma \times R \times \{0, 1,..., M\}, R_T)$ where M is the maximal tail-rank of symbols in Σ and R_T is defined as follows.

For $\sigma \in \Sigma_{0*}$

$$\sigma \to \langle \sigma, \gamma, 0\rangle (\langle p_1, 1\rangle \cdots \langle p_m, m\rangle) \text{ is in } R_T$$
$$\text{if } \gamma\text{:} \quad \sigma \to \sigma'(p_1 \cdots p_m) \text{ is in } R.$$

For $\sigma \in \Sigma_{1m}$,

$$[\langle q, i\rangle]\, \sigma \to \langle \sigma, \gamma, i\rangle (\langle p_1, 1\rangle \cdots \langle p_m, m\rangle) \text{ is in } R_T$$
$$\text{if } \gamma\text{:} \quad [q]\, \sigma \to \sigma'(p_1 \cdots p_m) \text{ is in } R.$$

For $\sigma \in \Sigma_{nm} (n > 1)$,

$$[\langle q_1, 1\rangle \cdots \langle q_n, 1\rangle]\, \sigma \to \langle \sigma, \gamma, 1\rangle (\langle p_1, 1\rangle \cdots \langle p_m, m\rangle) \text{ is in } R_T$$
$$\text{if } \gamma\text{:} \quad [q_1 \cdots q_m]\, \sigma \to \sigma'(p_1 \cdots p_m) \text{ is in } R.$$

The deterministic top-down automaton A_1 to recognize L' is very similar to T. It has the same set of states as T and has a rule: $[\alpha]\langle \sigma, \gamma, i\rangle \to \langle \sigma, \gamma, i\rangle' [\beta]$ iff T has a rule $[\alpha]\, \sigma \to \langle \sigma, \gamma, i\rangle [\beta]$, where $\sigma \in \Sigma$, $\gamma \in R$, $i \in \{0, 1,..., M\}$ and α and β are (possibly empty) sequences of states of T.

The following automaton A_2 in $2D$ recognizes L'. It traverses an input d-dag in pre-order, checking the following properties:

(i) A root is labeled as $\langle \sigma, \gamma, 0\rangle$ for some $\sigma \in \Sigma$, $\gamma \in R$.

(ii) If a node $\langle \sigma, \gamma, i\rangle$ has more than one father, then $i = 1$.

(iii) When a current node is $\langle \sigma, \gamma, i\rangle$ in Figure 23(a), it visits all its sons and checks their labels as in the figure and that the right-hand side of γ is $\sigma'(q_1 \cdots q_m)$ and the left-hand side of γ_j is $[q_j]\, \tau_j$, $1 \leqslant j \leqslant m$ for some $q_1,..., q_m \in Q$.

(iv) When a current node is $\langle \sigma_j, \gamma_j, i_j\rangle$ in Fig. 23(b) for some $1 \leqslant j \leqslant n$, it visits its son and checks that the right-hand side of γ_j is $\sigma'(q_j)$ and the left-hand of γ' is $[q_1 \cdots q_m]\, \tau$ for some $q_1,..., q_m \in Q$.

The distinction between the cases (iii) and (iv) can be determined by simply making one trip to the first son of a current node. At the same time, in the case of (iv), it can compute j. The number i in $\langle \sigma, \gamma, i\rangle$ is used to determine the next node to be visited both in traversing the input d-dag and in checking property (iii). Thus A_2 accepts an input in which every node has these properties. The formal details are omitted.

Finally the relabeling h is defined by $h(\langle \sigma, \gamma, i\rangle) = \sigma$ for each $\langle \sigma, \gamma, i\rangle \in \Sigma \times R \times \{0,..., M\}$ to erase the extra information. ∎

Remark. The language L' defined in the proof is actually the set of all

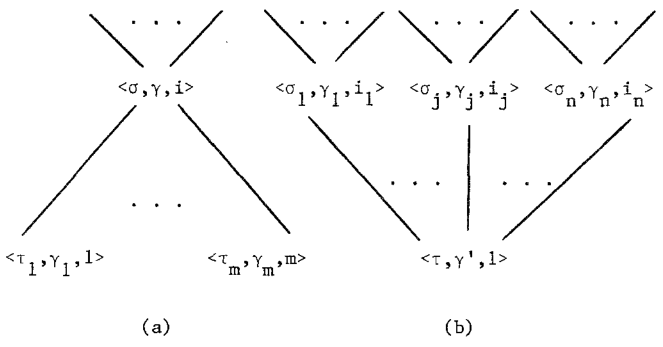


FIG. 23. Nodes in $T(L)$.

the derivation dags of the phrase-structure grammar which has all the rules $\langle\sigma, \gamma, i\rangle \rightarrow \langle\tau_1, i_1, 1\rangle \cdots \langle\tau_m, \gamma_m, m\rangle$ satisfying property (iii) in the proof and all the rules $\langle\sigma_1, \gamma_1, i_1\rangle \cdots \langle\sigma_n, \gamma_n, i_n\rangle \rightarrow \langle\tau, \gamma, i\rangle$ satisfying property (iv). ∎

7. CLOSURE PROPERTIES OF D-DAG LANGUAGES

As a result of the previous analysis of the recognition power of dag automata, we have obtained four different classes of d-dag languages. In this section we investigate the closure properties of these classes under the boolean operations and finite state relabelings.

First, all four classes are closed under intersection.

7.1. THEOREM. *All the classes $RECOG_D$, $\mathscr{L}(DT)$, $\mathscr{L}(2N)$ and $\mathscr{L}(2D)$ are closed under intersection.*

Proof. Let $A_1 = (Q_1, \Sigma, R_1)$ and $A_2 = (Q_2, \Sigma, R_2$ be two parallel automata of the same type. The automaton A_3 to recognize $L(A_1) \cap L(A_2)$ is constructed using the standard cartesian product construction.

On the other hand, in the case of dag-walking automata the "intersection automaton" is defined by applying the two automata one after the other. ∎

In Theorem 5.4, for each dag-walking automaton A in $2N-PD$, we constructed an equivalent deterministic dag-walking automaton $\hat{A}$ in $2D-PD$. We recall that $\hat{A}$ never blocks and the computation of $\hat{A}$ always terminates. Therefore, we can define A' in $2D-PD$ to recognize the complement of $L(\hat{A})$ by specifying the set of final states to be the complement of the set of final states of $\hat{A}$. Thus,

7.2. THEOREM. *$RECOG_D$ is closed under complementation.* ∎

Remark. Complementation of a d-dag language is taken with respect to the set of all d-dags over the alphabet in which the d-dag language is defined. ▮

In Theorems 5.5 and 5.6, we proved that the language L of Example 5.3 is not in $\mathscr{L}(2N)$, while its complement K is in $\mathscr{L}(2N)$.

7.3. COROLLARY. *$\mathscr{L}(2N)$ is not closed under complementation.* ▮

We consider next the operation of union.

7.4. THEOREM. *$RECOG_D$ and $\mathscr{L}(2N)$ are closed under union. $\mathscr{L}(DT)$ is not closed under union.*

Proof. Given two nondeterministic automata A_1 and A_2 of the same type, we can easily construct an automaton A_3 (of that type) which nondeterministically applies A_1 or A_2 to input d-dags.

Both of the singleton languages

$$\left\{\begin{array}{ccc} & s & \\ / & & \backslash \\ a & & b \end{array}\right\} \quad \text{and} \quad \left\{\begin{array}{ccc} & s & \\ / & & \backslash \\ b & & a \end{array}\right\}$$

are obviously in $\mathscr{L}(DT)$. However, by Theorem 3.5, their union is not in $\mathscr{L}(DT)$. ▮

The next result is immediate from Theorem 7.4 and de Morgan's law.

7.5. COROLLARY. *$\mathscr{L}(DT)$ is not closed under complementation.* ▮

The next theorem deals with transformations performed by finite state relabelings.

7.6. THEOREM. *$RECOG_D$ is closed under finite state relabelings.*

Proof. Let $L \in RECOG_D$ be a language over the alphabet Σ, A a parallel dag automaton to recognize L and let T be a finite state relabeling. Define T' from T by priming all the symbols of Σ in T. Looking upon A as a finite state relabeling $T(L) = \mathrm{ran}(A \circ T')$. Hence the theorem follows from Corollary 4.5 and Lemma 4.6. ▮

Theorems 6.3 and 7.6, together with the remark following Theorem 6.3, give the well-known result: $RECOG_D$ is the class of all the languages obtained by relabeling derivation dags of phrase-structure grammars.

Since a relabeling is a special kind of finite state relabeling, the next corollary is immediate from Theorem 6.3.

TABLE I

Closure Properties of the Classes of *d*-dag Languages

	$RECOG_D$	$\mathscr{L}(DT)$	$\mathscr{L}(2N)$	$\mathscr{L}(2D)$
Union	Yes	No	Yes	?
Intersection	Yes	Yes	Yes	Yes
Complement	Yes	No	No	?
Finite state relabeling	Yes	No	No	No

7.7. COROLLARY. *None of $\mathscr{L}(DT)$, $\mathscr{L}(2N)$ or $\mathscr{L}(2D)$ is closed under finite state relabelings.* ▮

Table I summarizes the closure properties of the families of *d*-dag languages. The problems of ,closure under union and complementation for $\mathscr{L}(2D)$ are open and we conjecture that the answers to both are negative.

8. RECOGNITION OF DOAGs

In this section, we discuss recognition of DOAGs by various dag automata introduced in this paper. As remarked after Definition 3.1, it is straightforward and needs no change of the definitions to apply parallel dag automata to DOAGs. The DOAG language accepted by a parallel dag automaton $A = (Q, \Sigma, R)$ is $L(A) = \{d \in G_\Sigma \mid d \vdash_A^* d'\}$. Also, definitions of configurations, computation step $\vdash_B$ and acceptance of an input by a dag-walking automaton $B = (Q, \Sigma, \Gamma, \delta, q_0, \gamma_0, F)$, given in Section 5 are valid for DOAGs. Thus the DOAG language accepted by B is $L(B) = \{d \in G_\Sigma \mid (q_0, d, (n_1), \gamma_0) \vdash_B^* (p, d, (\,), \lambda)$ with some $p \in F\}$; where n_1 is the root of DOAG d. Using the same notation as before, $\mathscr{L}(K) = \{L(A) \mid A \in K\}$ is the class of DOAG languages defined by automata in a class K of dag automata.

A number of results and their proofs concerning the power of dag automata to recognize languages hold unmodified for DOAGs.

8.1. THEOREM. *Theorems* 3.4, 3.5, 5.4, 5.5, 5.6, 6.1, *and* 6.2 *hold for DOAGs.*

Proof. Theorems 3.5, 5.5 and 5.6 were proved even for tree languages which can be regarded as DOAG languages. Proofs of other theorems are exactly the same as those given for *d*-dags. ▮

Thus all the results except Theorem 3.6 used to demonstrate Fig. 22 hold for DOAGs. However, Theorem 3.6 does not go through because DOAGs are not necessarily planar, a property which was crucial in the proof of that theorem. In fact, we can prove the following:

8.2. THEOREM. *When parallel dag automata operate on DOADs, nondeterministic bottom-up automata are more powerful than the deterministic ones.*

Proof. We define a set of DOAGs L as follows. Let t be a tree in which all the nodes except the leaves have exactly two sons, and which has an even number of leaves half of which are labeled by "c" and half by "d." The internal nodes are labeled by "a." Then define a DOAG d from t by adding k new nodes (all labeled by "b") which will be the leaves of d, where

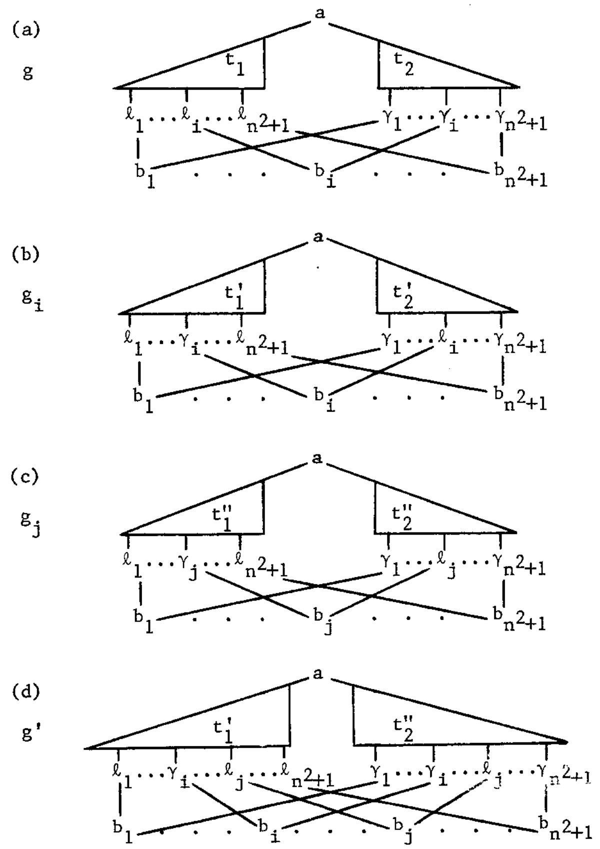

(a) g
a
t_1 t_2
$\ell_1 \dots \ell_i \dots \ell_{n^2+1}$ $\gamma_1 \dots \gamma_i \dots \gamma_{n^2+1}$
$b_1 \dots b_i \dots b_{n^2+1}$
(b) g_i
t'_1 t'_2
$\ell_1 \dots \gamma_i \dots \ell_{n^2+1}$ $\gamma_1 \dots \ell_i \dots \gamma_{n^2+1}$
(c) g_j
t''_1 t''_2
$\ell_1 \dots \gamma_j \dots \ell_{n^2+1}$ $\gamma_1 \dots \ell_j \dots \gamma_{n^2+1}$
(d) g'
t'_1 t''_2
$\ell_1 \dots \gamma_i \dots \ell_j \dots \ell_{n^2+1}$ $\gamma_1 \dots \gamma_i \dots \ell_j \dots \gamma_{n^2+1}$
$b_1 \dots b_i \dots b_j \dots b_{n^2+1}$

FIG. 24. DOAGs g, g_i, g_j and g'.

$k = |\text{leaves}(t)|/2$. Each now "b" node is connected to one "c" and one "d" node. The set L consists of all DOAGs obtained in this way.

L is recognizable by the following automaton B in NB:

$$(\{p, q, r\}, \{a, b, c, d\}, \{b \to [pq]\, b', b \to [qp]\, b', c(p) \to [r]\, c',$$
$$d(q) \to [r]\, d', a(rr) \to [r]\, a', a(rr) \to a'\}).$$

On the other hand, no automaton in DB can recognize L. Let $\Sigma = \{a, b, c, d\}$ and suppose that an automaton $A = (Q, \Sigma, R)$ in DB regonizes L. Let n be the cardinality of Q and let g be the DOAG of Fig. 24(a). g has $n^2 + 1b$'s, each of which, say b_i, is connected to a leaf of the subtree t_1 (labeled by ℓ_i) and to a leaf of the subtree t_2 (labeled by γ_i) $1 \leqslant i \leqslant n^2 + 1$. Now we define a DOG g_i by exchanging the labels of fathers of b_i in g, $1 \leqslant i \leqslant n^2 + 1$, see Fig. 24(b) and (c). Obviously g_i is in L, and therefore is accepted by A. Since for the symbol $a \in \Sigma_{02}$ there are at most n^2 applicable rules of A, two DOAGs, say g_i and g_j, must be accepted by A by applying the same rule at their roots. Let this rule be $a(pq) \to a'$, $p, q \in Q$. Now, define another DOAG g' from g by changing the label of the left father of b_i into γ_i and the label of the right father of b_j into ℓ_j, see Fig. 24(d). Thus, $g' \notin L$. Due to determinism, however, A must process the subtree t_1' (t_2'' respectively) of g' in precisely the same way as it does it in g_i (g_j). Thus, A arrives at the root of g' in state p (from the left) and q (from the right), thereby accepting g' by the rule $a(pq) \to a'$. This is a contradiction. ∎

Note that the associated automaton (see the proof of Theorem 3.4) with the automaton B of this proof is in DT. Therefore, automata in DT are incomparable to those in DB in their power of recognizing DOAGs. Figure 25 shows the inclussion diagram of classes of DOAG languages definable by various dag automata.

$$\begin{tikzcd}
& \mathcal{L}(\mathrm{NB}) = \mathcal{L}(\mathrm{NT}) \arrow[dl, dash] \arrow[dr, dash] & \\
\mathcal{L}(\mathrm{DT}) & & \mathcal{L}(\mathrm{DB}) = \mathcal{L}(\mathrm{2N\text{-}PD}) = \mathcal{L}(\mathrm{2D\text{-}PD}) \arrow[d, dash] \\
& & \mathcal{L}(\mathrm{2N}) \arrow[d, dash] \\
& & \mathcal{L}(\mathrm{2D})
\end{tikzcd}$$

FIG. 25. Inclusion diagram of DOAGs.

ACKNOWLEDGMENT

We would like to thank the referee of the paper whose constructive and detailed comments have helped us to improve both the accuracy and the style of the paper.

RECEIVED: March 12, 1980; REVISED: June 3, 1981

REFERENCES

ARBIB, M. A. AND GIVE'ON, Y. (1968), Algebra automata I: Parallel programming as a prelogomena to the categorical approach, *Inform. Contr.* **12**, 331–345.

AHO, A. V. AND ULLMAN, J. D. (1970), Transformations on straight line programs, *in* "Proceedings of the 2nd Annual ACM Symposium on Theory of Computing," Northampton, Mass., pp. 136–148.

AHO, A. V. AND ULLMAN, J. D. (1971), Translations on a context-free grammar; *Inform. Contr.* **19**, 439–475.

BAKER, B. S. (1973), "Tree Transductions and Families of Tree Languages," Ph.D. thesis, Harvard University.

BLUM, M. AND HEWITT, C. (1976), Automata on a two-dimensional tape, *in* "Proceedings of the 9th Symposium on Switching and Automata Theory," pp. 155–160.

BUTTELMANN, H. W. (1975), On the syntactic structure of unrestricted grammars, *Inform. Control.* **29**, 29–101.

DONER, J. E. (1979), Tree acceptors and some of their applications, *J. Comput. System Sci.* **4**, 406–451.

ENGELFRIET, J. (1975a), "Tree Automata and Tree Grammars," Lecture Notes DAIMI FN–10, University of Aarhus, Denmark.

ENGELFRIET, J. (1975b), Bottop-up and top-down tree transformations—A comparison, *Math. Systems Theory* **9**, 193–231.

ENGELFRIET, J. (1977), Top-down tree transducers with regular look-ahead, *Math. Systems Theory* **10**, 289 303.

ENGELFRIET, J., ROZENBERG, G., AND SLUTZKI, G. (1978), Tree transducers, *L*-systems, and two-way machines, *in* "Proceedings of the 10th annual ACM Symposium on Theory of Computing," San Diego, California, pp. 66-74.

HART, J. M. (1974), Acceptors for the derivation languages of phrase-structure grammars, *Inform. Contr.* **25**, 75–92.

HART, J. M. (1975a), The derivation language of a phrase-structure grammar, *J. Comput. System Sci.* **12**, 64–79.

HART, J. M. (1975b), Derivation language and syntactic categories, *Inform. Contr.* **28**, 204–220.

HOPCROFT, J. E. AND ULLMAN, J. D. (1967), Nonerasing stack automata, *J. Comput. System Sci.* **1**, 166–186.

KAMIMURA, T. (1979), "Automata on Directed Acyclic Graphs," Ph.D. dissertation, University of Delaware.

KAMIMURA, T. AND SLUTZKI, G. (1979), "Parallel and Two-Way Recognizers of Directed Acyclic Graphs," MFCS '79, Olomouc, Czechoslovakia; Lecture Notes in Computer Science No. 74, pp. 317–325, Springer-Verlag, Berlin/New York/Heidelberg.

MILGRAM, D. L. (1975), Web automata, *Inform. Contr.* **29**, 162–184.

MYLOPOULOUS, J. (1972), On the relation of graph grammars and graph automata, *in* "Proceedings of the 13th Symposium on Switching and Automata Theory," pp. 108–120.

ROUNDS, W. C. (1970), Mappings and grammars on trees, *Math. Systems Theory* **4**, 257–287.

SHEPERDSON, J. C. (1959), The reduction of two-way automata to one-way automata, *IBM J. Res. Develop.* **3**, 198–200.

THATCHER, J. W. (1973), Tree automata; informal survey, *in* "Currents in the Theory of Computing" (A. V. Aho, Ed.), Prentice–Hall, Englewood Cliffs, N. J., pp. 143–172.

THATCHER, J. W. AND WRIGHT, J. B. (1968), Generalized finite automata theory with an application to a decision-problem of second order logic, *Math. Systems Theory* **2**, 57–81.