Properties that Characterize LOGCFL*

H. Venkateswaran
School of Information and Computer Science
Georgia Institute of Technology, Atlanta, Georgia 30332-0280

ABSTRACT

Two properties, called *semi-unboundedness*, and *polynomial proof-size*, are identified as key properties shared by the definitions of LOGCFL on several models of computations. The semi-unboundedness property leads to the definition of semi-unbounded fan-in circuit families. These are circuits obtained from unbounded fan-in circuits by restricting the fan-in of gates of one type. A new characterization of LOGCFL is obtained on such a model in which the fan-in of the AND gates are bounded by a constant. This property also suggests new characterizations of LOGCFL on the following models: alternating Turing machines [3], nondeterministic auxiliary pushdown automata [5], and bounded fan-in Boolean circuits [6].

*Most of this work was supported by the National Science Foundation under grants DCR-8301212 and DCR-8352093 while the author was at the University of Washington, Seattle. A preliminary version of this paper was presented at the nineteenth annual Symposium on Theory of Computing, New York, 1987.

1 Introduction

LOGCFL is the class of languages log space reducible to context-free languages. It is a parallel complexity class known to contain a number of natural problems including non-context-free languages [7, 9, 15, 17, 18]. Membership in LOGCFL implies the existence of space-efficient and fast parallel recognition algorithms for the languages in the class [13, 14]. The class LOGCFL has been well studied from a complexity theory point of view in that complete problems have been identified for this class [6, 17], and characterizations of the class on several models of computation are known [6, 11, 14, 19, 23]. This paper carries the work on LOGCFL further by studying the structure of the computations in this class. Specifically, we identify two key properties, called *semi-unboundedness*, and *polynomial proof-size* for several models of computations on which LOGCFL has been characterized. The semi-unboundedness property leads to the definition of new models of computation based on unbounded fan-in circuits. These are circuits obtained from unbounded fan-in circuits by restricting the fan-in of gates of one type. We obtain a new characterization of LOGCFL on such a model in which the fan-in of the AND gates are bounded by a constant. In this paper, we will only consider unbounded fan-in circuits in which the AND gates have bounded fan-in and refer to them as *semi-unbounded* fan-in circuits. This assumption is not quite restrictive because Borodin et al. [2] have recently shown that families of size $C(n)$ and depth $D(n) \geq \log C(n)$ semi-unbounded fan-in circuits and families of size $C^{O(1)}(n)$ and depth $O(D(n))$ unbounded fan-in circuits in which the OR gates have bounded fan-in define the same classes. We consider bounded fan-in AND gates for semi-unbounded fan-in circuits since unbounded fan-in for OR gates seems to be a more natural model of nondeterminism. The semi-unboundedness property also suggests new characterizations of LOGCFL on the following models: alternating Turing machines [3], nondeterministic auxiliary pushdown automata [5], and bounded fan-in Boolean circuits [6]. We also show that all these characterizations possess another property namely, the polynomial proof-size property. Informally, a proof for an input that is accepted by a machine is the subtree of the machine's computation tree that verifies the fact that this is an accepted input.

One of the objectives of this work is to study the relationship between the classes LOGCFL and AC^1. AC^1 is the class of languages accepted by (1) a concurrent read, concurrent write parallel RAM in polynomial hardware and $O(\log n)$ time or, equivalently, (2) an alternating Turing machine in space $O(\log n)$ and alternation depth $O(\log n)$ or, equivalently, (3) a uniform family of unbounded fan-in circuits of polynomial size and $O(\log n)$ depth [20]. Ruzzo [14] showed that LOGCFL $\subseteq AC^1$. Showing that the inclusion is proper is likely to be difficult, as it would separate $\mathcal{P}$ (the class of polynomial time recognizable languages) from DSPACE$(\log n)$, settling this long standing open problem in complexity theory. This is because of the following relationship among these complexity classes [6]: DSPACE($\log n$) $\subseteq$ LOGCFL $\subseteq$ AC^1 $\subseteq$ NC $\subseteq$ $\mathcal{P}$. Here NC denotes the class of problems solvable in poly-log time by a parallel computer with a polynomial number of processors (see, for example, [6]).

By defining a resource called tree-size for alternating Turing machines, Ruzzo [14] characterized LOGCFL in this model as the class of languages accepted in $\log n$ space and polynomial tree-size. The use of different measures of resources (space and tree-size versus space and alternations) in their definitions make it difficult to understand how the classes LOGCFL and AC^1 differ. For instance, a polynomial tree-size bounded computation can have a polynomial number of alternations. Although Ruzzo [14] showed that $O(\log n)$ alternations suffice for such computations, thus showing that LOGCFL $\subseteq$ AC^1, it was not clear how these classes differed. Our results show that the difference between the two classes is the way the universal moves are used for their computations in the alternating Turing machine model: LOGCFL uses bounded universal moves and AC^1 uses unbounded universal moves. The characterization of LOGCFL using semi-unbounded fan-in circuits also demonstrates that the difference between the two classes can be quantified in terms of the fan-in of the AND gates in the unbounded fan-in circuit model: LOGCFL uses bounded fan-in AND gates and AC^1 uses unbounded fan-in AND gates. These characterizations provide more evidence supporting our belief that the inclusion of LOGCFL in AC^1 is proper.

Another objective of this work is to unify the many characterizations of LOGCFL using the notion of polynomial size proofs. This property is inherent in many of the previous characterizations of LOGCFL such as Ruzzo's characterization in terms of polynomial tree-size on alternating Turing machines. Sudborough [19] showed that LOGCFL is the class of languages accepted by nondeterministic auxiliary pushdown automata in $\log n$ space and polynomial time. We define the notion of a proof tree for this model, and show that LOGCFL is the class of languages accepted by nondeterministic auxiliary pushdown automata in $\log n$ space and polynomial tree-size. Similarly, by defining proofs in Boolean circuits in a natural way, we show that LOGCFL is the class of languages accepted by a uniform family of polynomial size Boolean circuits within polynomial proof-size. Thus, our contribution is to identify polynomial proof-size as a more general property that is shared by many definitions of LOGCFL. This gives a robust definition of the class LOGCFL. Also, showing polynomial proof-size for a problem seems to be an effective way of showing that it has an NC algorithm since LOGCFL $\subseteq$ NC^2.

As a final note, the semi-unbounded fan-in circuit model proposed in this paper has proved to be a useful combinatorial framework for studying the closure under complementation of the class LOGCFL. Recently, Borodin et al. [2] showed that the class LOGCFL is closed under complement using the semi-unbounded fan-in circuit characterization of LOGCFL presented in this paper.

This paper is organized as follows. Section 2 contains the characterization theorem in terms of the semi-unboundedness property. The characterization in terms of proof-size is presented in section 3. A discussion of semi-unbounded circuits is presented in section 4. Section 5 contains some of the interesting questions raised by this work.

2 Semi-unboundedness Characterization of LOGCFL

This section contains the theorem that provides the characterizations of LOGCFL using the semi-unboundedness property. The characterization will be shown for the following four models: alternating Turing machines, nondeterministic auxiliary pushdown automata (NAuxPDAs), bounded fan-in Boolean circuits, and semi-unbounded fan-in Boolean circuits. The characterization of LOGCFL and AC^1 by Venkateswaran and Tompa [23] using a two-person pebble game model of synchronous parallel computation motivated the discovery of the notion of semi-unboundedness.

Section 2.1 contains definitions and section 2.2 contains the main characterization theorem.

2.1 Definitions

We now define the different models and the parameters of each that will capture the notions of semi-unboundedness and polynomial proof size.

2.1.1 Boolean Circuits

A *Boolean circuit* G_n with n inputs is a finite acyclic directed graph with vertices having indegree zero or at least two and labelled as follows. Vertices of indegree zero are called *circuit-inputs* and are labelled from the set $\{0, 1, x_1, x_2, \ldots, x_n, \overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$. All other vertices (also called *gates*) are labelled either AND or OR. Vertices with outdegree zero are called *outputs*. The evaluation of G_n on inputs of length n is defined in the standard way. Typically, we will consider only circuits with one output vertex. This makes it convenient to consider circuits as language acceptors.

The language L_n accepted by a Boolean circuit G_n is defined as follows:

$L_n = \{x \mid x \text{ has length } n \text{ and } G_n \text{ evaluates to 1 on input } x\}.$

The *size* $C(G_n)$ of a circuit G_n is the number of edges in G_n. The *depth* of a vertex v in a circuit is the length of a longest path from any input to v. The depth of a circuit is the depth of its output vertex.

We have defined Boolean circuits without negation gates, since there is a well known technique to simulate, with a doubling of size and no increase in depth, a Boolean circuit with negations by a Boolean circuit in which the negations appear only at the inputs. (See, for example, [10].)

A *family* of circuits is a sequence $\{G_n \mid n = 0, 1, 2, \dots\}$, where the n-th circuit G_n has n inputs. The language L accepted by a family $\{G_n\}$ of circuits is defined as follows: $L = \cup_{n \geq 0} L_n$, where L_n is the language accepted by the n-th member G_n of the family. We will define uniformity for circuit families later.

The notions of alternation, accepting subtrees, and tree-size for Boolean circuits can be defined by analogy to the corresponding notions for alternating Turing machines. These definitions follow those by Venkateswaran and Tompa [23] for bounded fan-in Boolean circuits.

Alternation: A Boolean circuit G_n is said to have *alternation* bound $A(n)$ if, for all paths p in G_n from some input of G_n to the output gate, the number of edges on p connecting an AND gate to an OR gate or vice versa is at most $A(n)$.

Accepting Subtrees: Accepting subtrees are defined for Boolean circuits by considering the *tree-equivalent* $T(G)$ of a circuit G obtained by modifying G, by the construction below, so that every vertex in $T(G)$, except its output, has outdegree one, and $T(G)$ accepts the same language as G.

Given a circuit G, an inductive construction of $T(G)$, the tree-equivalent of G, is given below. Let V be the set of vertices in G and E be the set of edges. The construction is done in d stages where d is the depth of G.

Assume that at the start of stage $l \geq 1$, all vertices at depth $< l - 1$ have outdegree one. Note that this condition holds trivially at the start of stage one. After this stage of the construction, all vertices at depth $< l$ will have outdegree one. For all vertices v at depth $l - 1$, if v has outdegree $k > 1$ do the following. Let $u_1, \cdots, u_k$ be the vertices in V such that (v, u_i), $(1 \leq i \leq k)$, are the outgoing edges from v. Replace v by k vertices $v^{(1)}, \cdots, v^{(k)}$. Replace the subtree rooted at v by k copies, one rooted at $v^{(i)}$ for each i, $1 \leq i \leq k$. Replace the edge (v, u_i) by the edge $(v^{(i)}, u_i)$, $(1 \leq i \leq k)$.

The fact that $T(G)$, so constructed, accepts the same language as G is a consequence of the following claim: for any gate v in G, the gate v evaluates to one on x if and only if all its copies in $T(G)$ evaluate to one on input x. The proof of this claim is by induction on the depth of v coupled with the following observation: for any gate v in G with $d(v) \geq 1$, if v has $v_1, \dots, v_k$, for $k \geq 2$, as its immediate predecessors in G, then any copy v' of v in $T(G)$ has as its immediate predecessors a copy of $v_1, \dots, v_k$.

Let $x \in L$. An *accepting subtree* H of a circuit G on input x is a subtree of $T(G)$, its tree-equivalent, defined as follows:

- H includes the output gate,
- for any AND gate v included in H, all the immediate predecessors of v in $T(G)$ are included as its immediate predecessors in H,
- for any OR gate v included in H, exactly one immediate predecessor of v in $T(G)$ is included as its only immediate predecessor in H, and
- any vertex of indegree zero included in H has value one as determined by the input x.

The following fact is easy to verify.

Fact 1 A Boolean circuit G evaluates to one on input x if and only if there exists an accepting subtree of G on input x.

Tree-size: The notion of tree-size for Boolean circuits defined below is analogous to the notion of tree-size for alternating Turing machines defined by Ruzzo [14].

Let x be a length n input on which G_n evaluates to one. G_n is said to use tree-size $Z(n)$ on input x if there is an accepting subtree of G_n of size at most $Z(n)$. If for every $x \in L$, G_n uses tree-size $Z(n)$ on input x, then G_n is said to have *tree-size* $Z(n)$.

2.1.2 Semi-Unbounded Fan-in Boolean Circuits

A family of Boolean circuits is said to have semi-unbounded fan-in if there exists a constant $c > 0$ such that in any circuit in the family, all AND gates have fan-in at most c. Note that there are no restrictions on the fan-in of the OR gates in such circuits.

At the expense of doubling the size and depth, it will be assumed that for any OR (AND) gate v, v does not have as input another OR (AND) gate. Thus, depth in such a circuit is the same as its alternation depth.

To define uniformity for a family of semi-unbounded fan-in circuits, define the *direct connection language* L_{DC} of the family to be the set of strings of the form $< n, g, y >$ such that in G_n either (i) g and y are gate names and g is an input to the gate y, or (ii) g is a gate name and y is the type of the gate g, that is, y is one of AND, or OR, or a circuit-input or its negation. A gate name here is a binary string. It will be assumed that each vertex in the circuit G_n has a numbering such that the vertex numbers coded in binary have length $O(\log C(G_n))$, where $C(G_n)$ is the size of the circuit.

Define a family $\{G_n\}$ of semi-unbounded fan-in circuits of size $C(G_n)$ to be *uniform* if the corresponding direct connection language can be recognized by a deterministic Turing machine in space $O(\log C(G_n))$. This idea of uniformity based on a direct connection language for a Boolean circuit is adapted from Ruzzo's definitions [15]. This uniformity condition for families of semi-unbounded fan-in circuits is chosen for convenience. It should be noted that there are other uniformity conditions such as NC^1-uniformity [6] that can be used here without changing the results.

2.1.3 Bounded Fan-in Boolean Circuits

A family of Boolean circuits is said to have bounded fan-in if there exists a constant $c > 0$ such that in any circuit in the family, all gates have fan-in at most c. Without loss of generality, it will be assumed that the fan-in of all the gates in a bounded fan-in circuit is two.

We will use the following definition of uniformity, referred to as log-space uniformity in the literature (see, for example, [15]) for bounded fan-in Boolean circuits.

The *standard encoding* $\overline{G_n}$ of a Boolean circuit G_n is the string of 4-tuples of the form $[g, t, g_L, g_R]$ where vertex number g's left (right) input is the output of vertex number g_L (g_R) and t denotes the type of the vertex number g, that is, t is either AND or OR or *input*. For a vertex number g which is a circuit-input, g_R is omitted and g_L denotes whether the input is negated or not.

A family $\{G_n\}$ of circuits of size $C(G_n)$ is said to be *uniform* if there is a deterministic Turing machine that, on input 1^n, can generate the standard encoding $\overline{G_n}$ of G_n using $O(\log C(G_n))$ space.

Ruzzo[15] has shown that the log-space uniformity condition for bounded fan-in Boolean circuits is equivalent to the uniformity condition based on recognition of the corresponding direct connection language.

2.1.4 Nondeterministic Auxiliary Pushdown Automata

A nondeterministic auxiliary pushdown automaton (referred to as NAuxPDA) is a nondeterministic Turing machine with an additional worktape that is constrained to operate as a pushdown store. The space used on the pushdown tape is not counted in the space bound of the machine. A more formal definition is given by Cook [5].

Let M be a $\log n$ space bounded NAuxPDA and L be the language accepted by it. The following assumptions will be made about the machine.

- M accepts by entering a unique final state with its stack empty.
- There is a symbol $\perp$ that marks the bottom of the stack. The last move that M makes when it accepts is to pop this symbol $\perp$ from the stack. This pop move of M on an input that it accepts will be referred to as the *final* pop move of M on that input.
- M pushes and pops in units of $\log n$ bits, where n is the length of its input. (An exception to this convention is when M pops the symbol $\perp$ form the stack.) This can be arranged as follows [14]. M maintains a work tape to keep the top K bits of the stack where $0 \leq K \leq 2\log n$. If this work tape is empty (i.e. $K = 0$) and the stack is not empty, M pops the top $\log n$ bits from the stack onto it. If the worktape contains $2\log n$ bits, M pushes a block of $\log n$ bits (the bottom half of the $2\log n$ bits on the work tape) from it onto the stack. For other push or pop moves, M writes or reads these bits on the work tape. Henceforth, a push (pop) move of M will refer to pushing (popping) $\log n$ bits from the stack.
- M behaves deterministically while making a push or pop move.

Type of a Configuration: A configuration of M is of type push (pop, existential resp.) if M makes a push (pop, existential resp.) move from it.

Realizable Pairs: A pair (p, q) of configurations is called *realizable with respect to a computation sequence* of M if p precedes q in the sequence, the stack height in p and q are the same, and the stack level between p and q never goes below that in p. A pair (p, q) of configurations is called *realizable* if it is realizable with respect to some computation sequence of M [14, 5].

Mates: Given $x \in L$, let $A(x)$ be an accepting computation sequence of M on input x. For every push configuration p in $A(x)$ there is a balancing pop configuration in $A(x)$, namely the first pop configuration q succeeding p in the sequence $A(x)$ such that the successor p' of p in $A(x)$ and q form a realizable pair in $A(x)$. The configurations p and q will be referred to as *mates* in the sequence $A(x)$.

Accepting Trees: An accepting computation sequence of M on an input $x \in L$ can be viewed as a path $P(x)$ whose vertices are labeled with configurations of M and whose edges correspond to the move relation of M. Thus, its start vertex is labelled with the initial configuration of M and its end vertex is labelled with the final accepting configuration. A vertex labelled with an existential (push, pop resp.) configuration will be referred to as an *existential* (*push, pop* resp.) vertex. The

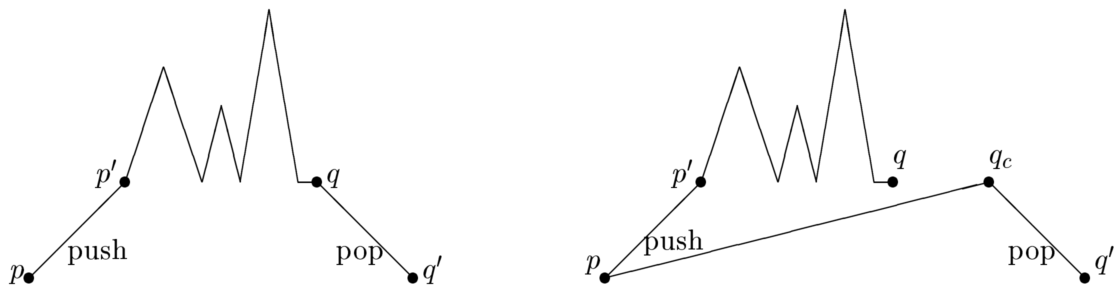

Figure 1: Tree associated with an accepting sequence

immediate successor of an existential vertex q in $P(x)$ is a vertex labelled with a configuration r such that there is a transition of M from q to r. The immediate successor of a push vertex p in $P(x)$ is a vertex labelled with the unique configuration reachable from p by the push move of M. The immediate successor of a pop vertex q in $P(x)$ is a vertex labelled with a configuration, say q', reachable from q by a pop move of M. Note that the stack information in q' is the same as the stack information in the push configuration that is the mate of q. Therefore, to construct the label of the immediate successor of a pop vertex, we need the stack information from its mate.

Given $x \in L$, let $P(x)$ be such an accepting path. Consider the tree obtained by converting $P(x)$ as follows. Let p be a push vertex, p' be its immediate successor and q be its mate pop vertex. Let the part of the accepting path $P(x)$ from the vertex p' to the vertex q be converted into a subtree by induction. With q associate a copy q_c, delete the edge from q to its successor q', and instead add an edge from q_c to q', and add an edge from p to q_c (see figure 1). This tree associated with $P(x)$ will be referred to as an *accepting tree* on input x.

For every $x \in L$, there is such an accepting tree $T(x)$. It should be noted that a pop vertex q occurs only as a leaf in such a tree and therefore has no successors in the tree, whereas the copy of a pop vertex q always has a successor in the tree.

It is also worth noting that $T(x)$ contains all the vertices of $P(x)$ and in addition it has a copy of every pop vertex in $P(x)$. Therefore, $|P(x)| < |T(x)| < 2|P(x)|$. That is, if M runs in time t then $T(x)$ has $O(t)$ vertices.

Tree-size: M is said to be $Z(n)$ tree-size bounded on input $x \in L$ of length n if there is an accepting tree on input x with $O(Z(n))$ vertices. M is said to be $Z(n)$ tree-size bounded if it is $Z(n)$ tree-size bounded over all inputs $x \in L$ of length n. By the observation in the preceding paragraph, tree-size and time of M are polynomially related.

Alternations: Let $x \in L$ and $T(x)$ be an accepting tree of M on input x. Define the number of *alternations* along a path of this tree to be the number of edges along the path connecting a non-existential vertex to an existential vertex or vice versa. M is said to be $A(n)$ alternation bounded on input $x \in L$ of length n if there is an accepting tree on input x such that the number of alternations over any path of this tree does not exceed $A(n)$. M is said to be $A(n)$ alternation bounded if it is $A(n)$ alternation bounded over all inputs $x \in L$ of length n.

It should be remarked here that Ladner, Lipton and Stockmeyer [12] considered the effect of adding alternation to auxiliary pushdown automata. That model differs from the one considered here in that an alternating auxiliary pushdown automaton has also universal states and the alternation measure for such a machine is the standard measure of alternations between existential and universal moves.

2.1.5 Semi-unboundedness

An alternating Turing machine or NAuxPDA M is said to be *semi-unbounded*, if there exists a constant $c > 0$ such that for all x accepted by M, there is an accepting tree $T(M, x)$ such that along any path in $T(M, x)$, the number of vertices labelled with non-existential configurations between each pair of successive vertices labelled with existential configurations is at most c.

A family of bounded fan-in Boolean circuits is said to have the *semi-unboundedness* property if there is a constant $c > 0$ such that for any circuit in the family the number of AND gates between every pair of successive OR gates along any path in the circuit is at most c.

A semi-unbounded fan-in circuit has, by definition, the semi-unboundedness property.

2.1.6 Notations

ASPACE,ALTERNATIONS (S)$(S(n), A(n))$ is the class of languages accepted by a semi-unbounded alternating Turing machine that is $O(S(n))$ space and $O(A(n))$ alternation bounded.

NAuxPDA SPACE,ALTERNATIONS (S)$(S(n), A(n))$ is the class of languages accepted by a semi-unbounded NAuxPDA that is $O(S(n))$ space and $O(A(n))$ alternation bounded.

Unbounded USIZE,DEPTH (S)$(C(n), D(n))$ is the class of languages accepted by a uniform family of semi-unbounded fan-in circuits with size $O(C(n))$ and depth $O(D(n))$. It should be recalled that the depth of such circuits is the same as their alternation depth.

USIZE,ALTERNATIONS (S)$(C(n), D(n))$ is the class of languages accepted by a uniform family of bounded fan-in Boolean circuits with the semi-unboundedness property and with size $O(C(n))$ and alternation $O(D(n))$.

2.2 The Characterization Theorem

The theorem in this section can be generalized, in a straightforward manner, to space and alternation bounds greater than $\log n$ for the machine models, and to alternation and depth bounds greater than $\log n$ and size bounds greater than polynomial for the circuit models. But we will prove this theorem only for the specific bounds in its statement to focus on the class LOGCFL.

Theorem 2

$$\begin{aligned}
\text{LOGCFL} &\subseteq \text{ASPACE,ALTERNATIONS (S)}(\log n, \log n) && (1)\\
&\subseteq \text{USIZE,ALTERNATIONS (S)}(n^{O(1)}, \log n) && (2)\\
&\subseteq \text{Unbounded USIZE,DEPTH (S)}(n^{O(1)}, \log n) && (3)\\
&\subseteq \text{NAuxPDA SPACE,ALTERNATIONS (S)}(\log n, \log n) && (4)\\
&\subseteq \text{LOGCFL}. && (5)
\end{aligned}$$

Proof

(1) LOGCFL $\subseteq$ ASPACE,ALTERNATIONS (S)$(\log n, \log n)$.

This follows from Ruzzo's simulation [14] of a space and tree-size bounded alternating Turing machine by a space and time bounded alternating Turing machine, along with the characterization of LOGCFL as the class of languages accepted by alternating Turing machines with $O(\log n)$ space and polynomial tree-size [14]. Ruzzo's simulating machine is semi-unbounded and uses $O(\log n)$ space and $O(\log n)$ alternations.

(2) ASPACE,ALTERNATIONS (S)$(\log n, \log n) \subseteq$
USIZE,ALTERNATIONS (S)$(n^{O(1)}, \log n)$.

Let L be accepted by a semi-unbounded alternating Turing machine M in $S(n) = O(\log n)$ space and $A(n) = O(\log n)$ alternations. The following assumption about reading inputs by an alternating Turing machine made by Ruzzo [15] will be found convenient. M has a special read state and a special index tape to read its inputs as follows: whenever it enters the read state with a, i on its index tape, M halts, and furthermore accepts if and only if the ith input symbol is a. This assumption implies that M is in a normal form such that only one input symbol is read along any path in its computation tree. The simulation below requires that M be in such a normal form. By a counting argument similar to the one in the proof of containment (5) below, M has polynomial tree-size. Therefore, if we apply Ruzzo's simulation (used in (1) above) to M, the resulting machine has all the required properties. It is semi-unbounded, is in the required normal form, and accepts L in $O(\log n)$ space and $O(\log n)$ alternations.

Since M is $O(\log n)$ space bounded, its running time is at most $T(n) = n^a$ for some constant a. Also since M is semi-unbounded, there exists a constant $K > 0$ such that for all x accepted by M, there is an accepting tree $T(M, x)$ such that along any path in $T(M, x)$, the number of vertices labelled with non-existential configurations between every pair of successive vertices labelled with existential configurations is at most K.

The construction is analogous to the simulation by Ruzzo [15] of an alternating Turing machine by a uniform family of Boolean circuits.

For $0 \le t \le T(n)$, $0 \le r \le A(n)$, $0 \le s \le K$, and a configuration c of M using space $S(n)$, there is a gate in the circuit in one of the following forms:

- $[t, r, c]$, if c is an existential configuration,
- $[t, r, s, c]$, if c is a universal configuration,
- $[t, c]$, if c is a read configuration.

The first component t in a gate name is used to avoid cycles in the circuit. The second component r in a gate name of the form $[t, r, c]$ or $[t, r, s, c]$ is used to cut off paths that have too many alternations. The third component s in gate name of the form $[t, r, s, c]$ is used to cut off paths that are not semi-unbounded. The type of a gate corresponding to a configuration c is, in general, OR (AND) if c is existential (universal).

Let c_I be the initial configuration of M. If c_I is an existential (universal) configuration the output gate is $[0, 0, c_I]$ ($[0, 0, 0, c_I]$).

To complete the description of the circuit, the construction of the inputs of a gate have to be specified. Consider a gate $[t, r, c]$ or $[t, r, s, c]$ corresponding to a configuration c of the machine. If $t + 1 > T(n)$, it has only one input, namely the constant zero. Otherwise, its inputs are constructed from the set D of all configurations reachable by M in one move from c. There will be one input corresponding to each $d \in D$. For any $d \in D$, if d uses space $> S(n)$, then the corresponding input is the constant zero. For all other $d \in D$, if d is a read configuration with a, i on its index tape, then the corresponding gate is $[t + 1, d]$ which is true if and only if the i-th circuit-input has value a. For all other $d \in D$, there are two cases.

Case 1: (Construct the inputs to $[t, r, c]$) If the configuration d is of the same type as c, i.e. existential, the corresponding input to $[t, r, c]$ is a gate $[t + 1, r, d]$. If d is a configuration of the opposite type, i.e. universal, then the corresponding input to $[t, r, c]$ is $[t+1, r+1, 0, d]$ if $r+1 \leq A(n)$, and it is the constant zero otherwise.

Case 2: (Construct the inputs to $[t, r, s, c]$) If the configuration d is of the same type as c, i.e. universal, the corresponding input to $[t, r, s, c]$ is a gate $[t+1, r, s+1, d]$ if $s+1 \leq K$, and it is the constant zero otherwise. If d is a configuration of the opposite type, i.e. existential, then the corresponding input to $[t, r, s, c]$ is $[t+1, r+1, d]$ if $r+1 \leq A(n)$, and it is the constant zero otherwise.

The correctness of this construction, its analysis, and the fact that the resulting circuit is log space uniform all follow as in Ruzzo's simulation [15].

(3) USIZE,ALTERNATIONS (S)$(n^{O(1)}, \log n) \subseteq$
Unbounded USIZE,DEPTH (S)$(n^{O(1)}, \log n)$.

Let $\{G_n\}$ be a uniform family of bounded fan-in Boolean circuits with size $C(n) = n^a$ for some constant $a \geq 1$, and alternations $A(n) = O(\log n)$. Let $\{G_n\}$ satisfy the semi-unboundedness property. We will define a family $\{H_n\}$ of semi-unbounded fan-in circuits, that is, circuits with unbounded fan-in for OR gates and bounded fan-in for AND gates, such that $\{H_n\}$ and $\{G_n\}$ accept the same language. We will also show how each H_n can be constructed by a log-space Turing machine. The basic idea of the construction is to make as inputs to an OR (AND) gate all non-OR (non-AND) gates from which it is reachable in G_n through only OR (AND) gates.

The construction of the n-th member H_n of this family will now be described. Let $D(n) = \lceil \log_2 C(n) \rceil$.

Gates in the circuit H_n are all of the form $[c]$, or $[d, @]$, or $[c, d]$, or $[s, c, d]$, or $[s, c, d, e]$, where $0 \leq s \leq D(n)$, @ is a special symbol, and c, d and e are the names of gates or circuit-inputs in G_n. The output gate of H_n is $[r_0]$, where r_0 is the output of G_n. For any gate c, the type of the gate $[c]$ is the same as that of the gate c. Given a gate $[c]$, its inputs are defined as follows.

Case 1: $[c]$ is an OR gate. Its inputs are gates $[c, d]$ for all non-OR gates d in G_n. Each of the gates $[c, d]$ is an AND gate and it has two inputs $[0, c, d]$ and $[d, @]$ defined as follows.

- The gate $[0, c, d]$ is the output of a $D(n)$ depth semi-unbounded fan-in circuit that checks whether c is reachable from d in G_n using only OR gates. This subcircuit for testing reachability will be described later.
- The gate $[d, @]$ is an OR gate with a single input $[d]$ defined as follows. If d is a circuit-input of G_n, $[d]$ is the corresponding circuit-input of H_n. Otherwise, d is an AND gate. Then $[d]$ is also an AND gate. Its inputs are constructed recursively, using Case 2 below.

Case 2: $[c]$ is an AND gate. Let $d_1, d_2, \cdots, d_k$ be all the non-AND gates in G_n from which the gate c is reachable in G_n using only AND gates. Note that k is a constant since $\{G_n\}$ has the semi-unboundedness property. The inputs to $[c]$ are the gates $[d_i]$ for $1 \leq i \leq k$. For these gates, the type of the gate $[d_i]$ is OR if d_i is an OR gate, and it is a circuit-input of H_n if d_i is a circuit-input of G_n. The inputs to each of the OR gates $[d_i]$ are constructed recursively.

Reachability Subcircuit: Given a gate $[s, c, d]$ with $0 \leq s \leq D(n)$, the goal is to describe a subcircuit of which this gate is the output, such that the subcircuit checks that c is reachable from d in G_n by a path of at most $2^{D(n)-s}$ OR gates (see also the construction by Borodin [1]).

If d is an immediate predecessor of c in G_n, then $[s, c, d]$ is the constant one. Otherwise, if $s + 1 > D(n)$, then $[s, c, d]$ is the constant zero. Otherwise, the gate $[s, c, d]$ is an OR gate. Its inputs are gates $[s + 1, c, d, e]$ for all OR gates e in G_n. Each of the gates $[s + 1, c, d, e]$ is an AND gate, and it has the two inputs $[s + 1, c, e]$ and $[s + 1, e, d]$. These two subcircuits are constructed recursively.

The circuit H_n has polynomial size and $O(\log n)$ depth. Also, the OR gates in H_n may have polynomial fan-in whereas the fan-in of the AND gates is bounded by a constant.

It is easy to show that G_n and H_n accept the same language.

We will show the existence of a uniformity machine $U(H)$ that recognizes the direct connection language for the circuit family $\{H_n\}$ using $O(\log n)$ space.

- Checking the type of a gate of H_n: Given $< n, g, t >$, where t is a gate type, $U(H)$ can check whether the type of the gate g is t in $O(\log n)$ space as follows. If g is a gate of the form $[c]$, its type is the same as the gate c in G_n, and this can be verified using the uniformity machine of G_n. If g is a gate of the form $[c, d]$ or $[s, c, d, e]$, then its type is AND. If g is a gate of the form $[d, @]$, then its type is OR. Finally, if g is a gate of the form $[s, c, d]$, its type can be determined as follows. The machine $U(H)$ can check whether d is an input of c in G_n using the uniformity machine of G_n, and if this is the case verify that g is the constant 1. Otherwise, $U(H)$ can check in $O(\log n)$ space whether $s + 1 > D(n)$ and if this is the case verify that g is the constant 0. Otherwise, $U(H)$ can check whether the type t denotes OR.

- Checking the input of a gate of H_n: Given $< n, g, y >$, where g and y are gate names, $U(H)$ can check in $O(\log n)$ space whether g is an input of y as follows. The machine $U(H)$ first determines the types of the gates g and y as discussed above, using the uniformity machine of G_n if necessary.

- Suppose y is an OR gate $[c]$. Then $U(H)$ can check whether the gate g is an AND gate $[c,d]$ such that d is a non-OR gate of G_n. This latter condition can be verified using the uniformity machine of G_n.
- Suppose y is an AND gate $[c,d]$. Then $U(H)$ can check whether g is either $[0,c,d]$ or $[d,@]$.
- Suppose y is an OR gate $[d,@]$. Then $U(H)$ can check whether g is d and it is a non-OR gate of G_n. This latter condition can be verified using the uniformity machine of G_n.
- Suppose y is an OR gate $[s,c,d]$. Then $U(H)$ can check whether the gate g is an AND gate $[s+1,c,d,e]$ such that e is an OR gate of G_n. This latter condition can be verified using the uniformity machine of G_n.
- Suppose y is an AND gate $[s,c,d,e]$. Then $U(H)$ can verify that g is either $[s,c,e]$ or $[s,e,d]$.
- Suppose y is an AND gate $[c]$. Then $U(H)$ can check in $O(\log n)$ space whether g is a non-AND gate $[e]$ such that e is a gate of G_n of the same type as g, and such that the gate c is reachable from the gate e in G_n using only AND gates. It does this by doing a depth-first search from the gate c in G_n. A brief description of this search follows. Initially, the gate c is pushed onto a stack S for the purposes of backtracking. An input d of c is generated and it is verified with the uniformity machine of $\{G_n\}$ that d is an input of c. The type of d is also found using the uniformity machine of $\{G_n\}$. If d is of the same type as c, that is an AND gate, then d is pushed onto the stack S and an input of d is generated. This is repeated until a non-AND gate d_1 is found such that c is reachable from d_1 in G_n using only AND gates. If the gate d_1 is not the same as the gate e, and if the stack S is not empty, the AND gate at its top is popped and another non-AND gate found from which c is reachable using only AND gates. This process is repeated until either the gate e is found (as one of the non-AND gates from which c is reachable using only AND gates), or the stack S becomes empty. Since $\{G_n\}$ has the semi-unboundedness property, there are only a constant number of non-AND gates from which c is reachable in G_n using only AND gates. The stack S thus uses $O(\log n)$ space since gate names are short. The uniformity machine of $\{G_n\}$, which is used in the depth-first search, also uses $O(\log n)$ space, by assumption.

(4) Unbounded USIZE,DEPTH (S)$(n^{O(1)}, \log n) \subseteq$

NAuxPDA SPACE,ALTERNATIONS (S)$(\log n, \log n)$.

Let L be accepted by a uniform family $\{G_n\}$ of polynomial size semi-unbounded fan-in circuits with $O(\log n)$ depth. Given x of length n, a NAuxPDA M checks whether the circuit evaluates to one on x as described below.

Initially, M's stack is empty. M begins the simulation with the output gate r_0. Given a gate v and its type, M checks that v evaluates to one on x as follows. Let $C(v)$ denote the configuration of M as it begins checking the gate v.

Case 1: v is an OR gate. M existentially guesses one of its true inputs u and its type and verifies deterministically with the uniformity machine that the guesses are correct. It then recursively checks that the gate u evaluates to one.

Case 2: v is an AND gate. Then it has a constant number of, say k, inputs. M existentially guesses these inputs, say, $v_1, \cdots, v_k$, and their types and verifies with the uniformity machine that the guesses are correct. M then pushes the gates $v_2, \cdots, v_k$ onto the stack. Along with a gate its type is also pushed onto the stack. M then recursively checks that v_1 evaluates to one.

Case 3: v is a circuit-input. If its value is zero, M rejects. Suppose v has value one. M makes its final pop move and accepts if $\perp$ is at the top of the stack. Otherwise, M pops a gate u and its type from the stack and recursively checks that u evaluates to one.

Correctness: For correctness, it has to be shown that the output r_0 of the circuit G_n evaluates to one on input x if and only if M accepts starting from $C(r_0)$ and the top of the stack is the $\perp$ symbol.

Case 1: Let r_0 evaluate to one on input x. This case follows from the claim below since $\perp$ is at the top of the stack initially.

Claim: Let gate v evaluate to one on input x. Then there exists a pop configuration C' such that $(C(v), C')$ is realizable.

Proof of the Claim: This is by induction on the depth $d(v)$ of the gate v.

Basis: $(d(v) = 0)$ v is a circuit-input with value one. M immediately enters a pop configuration C'. Since M does not make push or pop moves to go from $C(v)$ to C', they form a realizable pair.

Induction hypothesis: Assume that the claim is true for all gates u at depth $d(u)$ with $0 \leq d(u) < d(v)$.

Induction: $(d(v) > 0)$ Suppose v is an OR gate. Since this evaluates to one on x, at least one of its inputs, say v_1, evaluates to one. The configuration $C(v)$ is an existential configuration and in checking v, M can move from $C(v)$ to $C(v_1)$ through only existential configurations. Since,

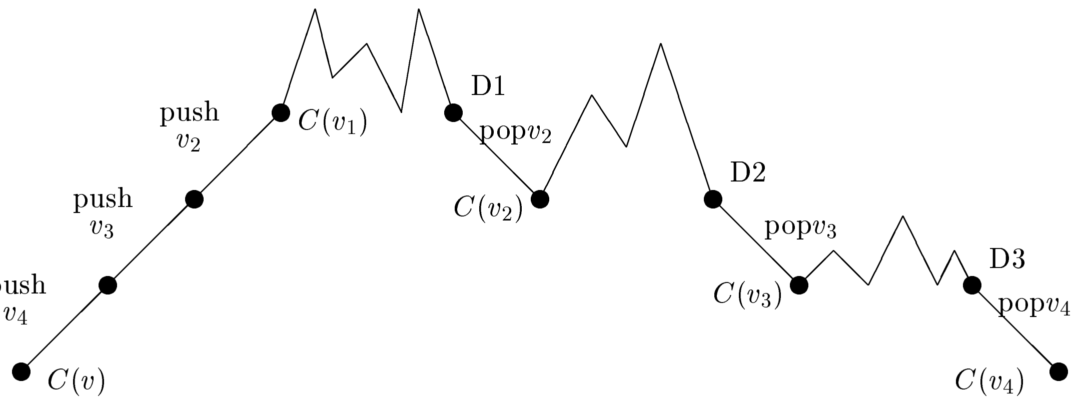

Figure 2: Simulating an *AND* with $k = 4$

by induction hypothesis, there exists a pop configuration C' such that $(C(v_1), C')$ is realizable, it follows that $(C(v), C')$ is also realizable.

Suppose v is an AND gate with inputs $v_1, \cdots, v_k$, for some constant k. Since v evaluates to one, all its k inputs evaluate to one. In checking v, M begins by existentially guessing these inputs and their types, verifying the guesses with the uniformity machine and pushing onto the stack the gates $v_2, \cdots, v_k$ and their types. M then recursively checks v_1. Therefore, $C(v)$ is an existential configuration, and M can move from $C(v)$ to $C(v_1)$ through a sequence of existential configurations followed by a sequence of $k-1$ push moves. Since $d(v_1) < d(v)$, by induction hypothesis, there exists a pop configuration D_1 such that $(C(v_1), D_1)$ is realizable. Now, the gate v_2 and its type is at the top of the stack at D_1. Let $C(v_2)$ be the configuration reached by M when it pops this gate name and its type. If $k > 2$, there is at least one more gate in the stack. In general, for $i < k$, $d(v_i) < d(v)$, and therefore, by induction hypothesis, there exists a pop configuration D_i such that $(C(v_i), D_i)$ is realizable. At D_i, the gate v_{i+1} and its type is at the top of the stack. Finally, at D_{k-1}, the gate v_k and its type will be at the top of the stack. Let $C(v_k)$ be the configuration reached by M when it pops this gate name and its type. It is easy to verify that $(C(v), C(v_k))$ form a realizable pair. (See figure 2.)

By induction hypothesis, there exists a pop configuration C' such that $(C(v_k), C')$ is realizable. Since, $(C(v), C(v_k))$ is realizable, it follows that $(C(v), C')$ is realizable. This proves the claim. □

Case 2: Let r_0 evaluate to zero on x. In general, for any gate v that evaluates to zero on input x, it will be shown that M rejects from $C(v)$. The proof is by induction on the depth $d(v)$ of v. The base case, with $d(v) = 0$, is clearly true. Assume that this is true for all gates u at depth $d(u)$ with $0 \leq d(u) < d(v)$.

Case 2.1: Let v be an OR gate. Now, the configurations corresponding to checking the inputs of v are reachable from $C(v)$ through only existential configurations, and all other configurations reachable from $C(v)$ through only existential configurations are rejecting. Since v evaluates to zero on x, all its inputs evaluate to zero. By the induction hypothesis, the corresponding configurations of M are all rejecting. Hence, $C(v)$ is rejecting.

Case 2.2: Let v be an AND gate with k inputs, $v_1, \cdots, v_k$. Since v evaluates to zero, one of these inputs evaluates to zero. Let i be the least index such that v_i evaluates to zero. If $i = 1$, v_1 evaluates to zero. By induction hypothesis, M rejects from $C(v_1)$. Since there is a unique sequence of existential configurations from $C(v)$ to $C(v_1)$, M rejects from $C(v)$. Let $i > 1$. Now, any computation sequence of M from $C(v)$ that does not reach $C(v_i)$ is rejecting. This is because, if M does not reach $C(v_i)$, then it does not reach a configuration from which v_i is popped, which implies that the stack will not become empty. But, any computation sequence that reaches $C(v_i)$ is also rejecting since, by induction hypothesis, $C(v_i)$ is rejecting. Therefore, M rejects from $C(v)$.

Analysis: Consider the space (exclusive of the stack) used by M on input $x \in L$ of length n. In checking a gate v, M must remember the gate v and its type. In checking an input v of the circuit, M must remember the input v, and the top of the stack which is either the symbol $\perp$ or a gate name and its type. This uses $O(\log n)$ space. If v is an OR gate, M has to record its true input, and its type. If v is an AND gate, M has to record all its inputs, at most k of them, and their types. Since k is a constant, M uses $O(\log n)$ space for all these. Finally, the uniformity machine is also $O(\log n)$ space bounded, so that the total space used by M is $O(\log n)$.

It only remains to show that M has $O(\log n)$ alternations and is semi-unbounded.

This is done by showing that, for all $x \in L$, M has an accepting tree $T(M, x)$ that can be divided into $O(\log n)$ layers such that each layer adds at most two alternations to any path from the root of $T(M, x)$ and that there is a constant number of non-existential vertices between successive existential vertices along any path in $T(M, x)$.

Since $x \in L$, there is an accepting subtree $T(G_n, x)$ of G_n on input x. Such an accepting subtree can be divided into $O(\log n)$ layers as follows.

Layer zero is the union of two singleton sets l_0 and e_0 each of which consists of the output OR gate. In general, given e_i, a set of OR gates or inputs in layer i of $T(G_n, x)$, layer $i+1$ is the union of two sets l_{i+1} and e_{i+1} defined as follows. Let $v \in e_i$ be an OR gate. v evaluates to one since it is in $T(G_n, x)$. Therefore, it has at least one non-OR input u that evaluates to one. The least, say, such u is included in l_{i+1}. If u is a circuit-input, then it has no predecessors in $T(G_n, x)$. If u is an AND gate then all its inputs are included in e_{i+1}. Note that these are OR gates or circuit-inputs of

G_n since G_n is a semi-unbounded fan-in circuit. Furthermore, these inputs of u themselves evaluate to one as they are in $T(G_n, x)$.

It is easy to see that, for any $u \in e_i$, the path in $T(G_n, x)$ from u to the output has at most i AND gates, and therefore, the construction above divides $T(G_n, x)$ into $O(\log n)$ layers.

An inductive construction of an accepting tree $T(M, x)$ of M corresponding to $T(G_n, x)$ on input $x \in L$ and a layering of $T(M, x)$ that parallels the layers of $T(G_n, x)$ will now be described.

The root of $T(M, x)$ is labelled $C(r_0)$. Layer zero is the union of two singleton sets L_0 and E_0 each of which consists of the root of $T(M, x)$. The existential vertex $C(r_0)$ in E_0 corresponds to the gate r_0 in e_0.

In general, layer i will be the union of two sets L_i and E_i. The construction of the tree will maintain the following invariant: E_i is the set of vertices that correspond to checking that the gates or inputs of G_n in e_i evaluate to one. Since $T(G_n, x)$ has $O(\log n)$ layers, this will ensure that $T(M, x)$ also has $O(\log n)$ layers.

Consider a gate or an input u in e_{i-1}. Let E_{i-1} contain a vertex labelled $C(u)$ corresponding to u.

Case 1: Suppose u is a circuit-input of G_n. Then its value is one. In checking u, M can move from $C(u)$ to a pop configuration C' through only existential configurations. Include in L_i the corresponding chain of existential vertices from the vertex labelled $C(u)$ to the vertex labelled C'. The vertex labelled C' has no successors in the tree.

Case 2: Suppose u is an OR gate. Since u is in $T(G_n, x)$, it evaluates to one. Therefore, there is a true non-OR gate v that is an input of u in $T(G_n, x)$. In checking the gate u, M can move from $C(u)$ to $C(v)$ through existential configurations. Let p be the corresponding chain of existential vertices with start vertex labelled as $C(u)$ and the end vertex labelled as $C(v)$. Include in L_i all the vertices in the chain p excepting the start and end vertices.

Case 2.1: Suppose v is an input of G_n. Then, by the layering of $T(G_n, x)$, v is in e_i. Include $C(v)$ in E_i.

Case 2.2: Suppose v is an AND gate with inputs $v_1, v_2, \cdots, v_k$ for some constant $k \geq 2$. Then, by the layering of $T(G_n, x)$, the inputs $v_1, v_2, \cdots, v_k$ of v are included in e_i.

Now $C(v)$ is an existential configuration. In checking v, M can move from $C(v)$ to P_k, a push configuration from which M pushes the gate v_k and its type onto the stack, through only existential configurations. Include in L_i the corresponding chain of vertices with start vertex labelled as $C(v)$ and the end vertex labelled as P_k. The vertex labelled P_k has two children: (see the definition of an accepting tree in section 2.1.4) a vertex labelled D_{k-1} corresponding to the mate of P_k and a vertex labelled P_{k-1} corresponding to the configuration reached by M by making the push move from P_k. These two children of P_k are included in L_i. Let $C(v_k)$ be the configuration reached by M from D_{k-1}. The configuration $C(v_k)$ corresponds to checking the gate v_k. Include the vertex labelled $C(v_k)$ in E_i.

If $k > 2$, P_{k-1} is a push configuration. In general, for a vertex labelled P_j, where $1 \leq j \leq k$, there are two children in L_i: a pop vertex labelled D_{j-1} corresponding to the mate of P_j and a vertex labelled P_{j-1} corresponding to the configuration reached by M by making the push move from P_j. Include these two children of P_j in L_i. Let $C(v_j)$ be the configuration reached by M from D_{j-1}. The configuration $C(v_j)$ corresponds to checking the gate v_j. Include the vertex labelled $C(v_j)$ in E_i. Finally, the vertex labelled P_1 has a single successor, namely the vertex labelled $C(v_1)$ corresponding to checking the gate v_1. Include $C(v_1)$ in E_i.

Therefore, for all $1 \leq j \leq k$, the set of vertices labelled $C(v_j)$ corresponding to checking the gate or input v_j is included in E_i. See figure 3 for a layer of an accepting tree corresponding to simulating an AND gate v with four inputs, v_1, v_2, v_3, and v_4.

From this construction, it is seen that at most k paths p_j branch from the vertex labelled P_k with the path p_j passing through the vertex labelled $C(v_j)$ in E_i. There is one alternation in going from $C(u)$ to P_k (since P_k is a push vertex and its predecessor is an existential vertex) and one alternation in going from P_k to $C(v_j)$ for $1 \leq j \leq k$ (since P_k is a push vertex and its successor $C(v_j)$ is an existential vertex). Therefore, in layer i at most two alternations are added along any path through it. Since the number of push or pop vertices between $C(u)$ and $C(v_j)$ for $1 \leq j \leq k$ is at most $k - 1$, M has the semi-unboundedness property. Since there are $O(\log n)$ layers in $T(M, x)$ and since each layer adds at most two alternations along any path through it, the number of alternations in $T(M, x)$ is also $O(\log n)$.

(5) NAuxPDA SPACE,ALTERNATIONS (S)$(\log n, \log n) \subseteq$ LOGCFL.

Let L be accepted by a semi-unbounded NAuxPDA M using $O(\log n)$ space and $O(\log n)$ alternations. It will be argued that for any $x \in L$, there is an accepting tree with polynomial size. This will show that M runs in polynomial time. The conclusion then follows from Sudborough's characterization of LOGCFL as the class of languages accepted by NAuxPDAs in polynomial time [19].

Given M and an input $x \in L$, we will construct an accepting tree $T(M, x)$ of M on x and show that it can be divided into $O(\log n)$ layers, just as in the argument for containment (4) above. Let $P(x)$ be the accepting sequence of M corresponding to the tree $T(M, x)$. Without loss of generality,

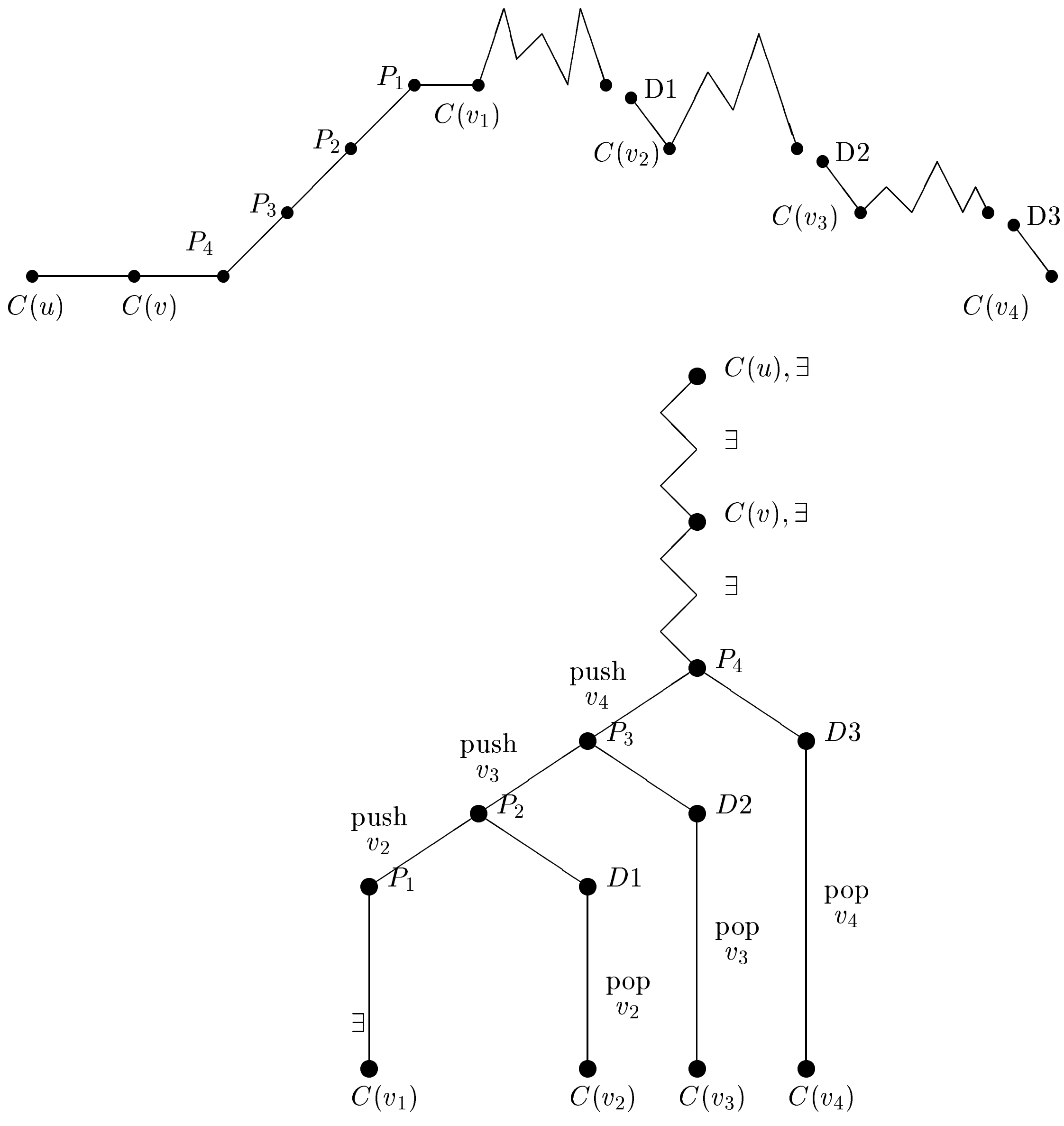

Figure 3: A Layer of an Accepting Tree with $k = 4$

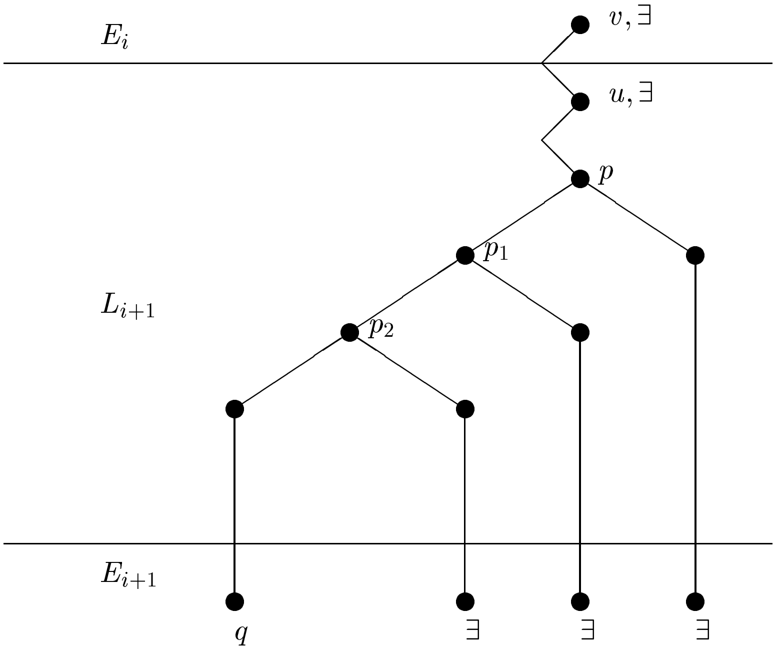

Figure 4: A Layer of the Tree with $l = 2$

it is assumed that the initial configuration is an existential configuration. It will also be assumed that a pop move does not immediately follow a push move, and that neither a push nor a pop move immediately follows a pop move. This will increase the number of alternations, but only by a constant factor since M is semi-unbounded.

Layer zero is the union of two singleton sets L_0 and E_0, each of which consist of the root of $T(M, x)$ corresponding to the initial configuration of M. Given the set E_i at layer i, the set of vertices at layer $i + 1$ is the union of the sets L_{i+1} and E_{i+1}, which are formed as follows.

For each vertex v in E_i, and each non-existential vertex u in $T(M, x)$ reachable from v through only existential vertices in $T(M, x)$, u is included in L_{i+1}. Let p be a push vertex so included in L_{i+1}. Then, between p and the next existential vertex q in $P(x)$, there are at most a constant number, say l, of push vertices $p_1, \cdots, p_l$. Consider the existential vertices immediately succeeding the mates of these l push vertices, the existential vertex immediately succeeding the mate of p, and the existential vertex q. This set, say S, of $l + 2$ existential vertices are included in E_{i+1}. There are $l + 2$ paths branching from p and the $l + 2$ existential vertices in S are the first existential vertices along these paths from p. (See figure 4.)

Therefore, there is at most one alternation along the path from p to any of these vertices in S. Since p is reachable from v only through existential vertices, there are at most two alternations along the path from v to any of these $l+2$ vertices. Since M has $O(\log n)$ alternations, it follows that $T(M,x)$ has $O(\log n)$ layers.

From the claim below, it follows that the total number of vertices in all the $O(\log n)$ layers is $n^{O(1)}$.

Claim: Let k be the maximum number of non-existential vertices between any two successive existential vertices along any path of $T(M,x)$. Then the number of vertices in layer i is at most $(k+1)^{i-1}n^{O(1)} + (k+1)^i$.

Proof of the Claim: It will be shown by induction on i that E_i has at most $(k+1)^i$ vertices and each of these form the start vertices of a polynomial size chain in layer $i+1$.

For the basis case, when $i=0$, E_0 has one vertex, namely, the root of $T(M,x)$. Assume that, for $i \geq 0$, E_i has at most $(k+1)^i$ vertices. By construction, all these are existential vertices. For each vertex $v \in E_i$, all vertices reachable from v using only existential vertices are in L_{i+1}. This corresponds to at most $(k+1)^i$ chains starting from each $v \in E_i$, either terminating in L_{i+1} or in a push vertex in L_{i+1}. Each of these chains have $n^{O(1)}$ vertices since M is $O(\log n)$ space bounded. From every such push vertex p, there are at most $k+1$ paths branching from p and the first existential vertices along these paths are in E_{i+1}. Thus, each chain contributes at most $k+1$ vertices to E_{i+1} and at most $n^{O(1)}$ vertices to L_{i+1}. Therefore, the number of vertices in layer $i+1$ is at most $(k+1)^i n^{O(1)} + (k+1)^{i+1}$. $\square$

Remarks: Theorem 2 suggests that the semi-unboundedness property captures a fundamental aspect of the computations in LOGCFL. This observation is further strengthened by considering other known characterizations of LOGCFL. We will briefly mention one such characterization. Immerman [11] used the size and number of variables needed to express properties in first order logic as resources in the model, and showed that LOGCFL is the class of properties expressible in $O(1)$ variables and $O(\log n)$ size when the universal quantifiers are restricted to be Boolean. By defining semi-unboundedness for this model in a way similar to the definitions in section 2.1.5, one can verify that this characterization of LOGCFL also possesses this property.

3 Polynomial Proof-size and LOGCFL

Proof Trees: An interesting concept that emerges from the definitions of LOGCFL on these models is that of a *proof tree*. Informally, a proof tree for a machine on an accepted input is a subtree of its computation tree that demonstrates the fact that this is an accepted input. We have defined in section 2.1 a notion of *accepting tree* for an input in three of our models: alternating Turing machines, Boolean circuits, and NAuxPDAs. In each case we can define a *proof tree* to be an accepting tree, and note that we can characterize LOGCFL in each of these models as the class of languages L for which there is a proof tree with polynomial size for each $x \in L$. We summarize this characterization in theorem 3 below.

Theorem 3 LOGCFL is the class of languages accepted by (a) alternating Turing machines and NAuxPDAs in $O(\log n)$ space and polynomial tree-size, (b) polynomial size Boolean circuits with polynomial tree-size.

The proof of this theorem follows from the characterization of LOGCFL in terms of alternating Turing machines by Ruzzo [14] and the proof of theorem 2 along with a counting argument similar to the one used for NAuxPDAs in that proof.

This theorem suggests that polynomial size proof trees characterize the computations in the class LOGCFL. This can be seen to hold for other models on which LOGCFL has been defined.

Based on the work of Skyum and Valiant [16], Cook [6] characterized LOGCFL as the class of languages accepted by a uniform family of Boolean circuits with polynomial size and polynomial degree. The degree of a Boolean circuit is the degree of the formal polynomial, over the two-element Boolean algebra, computed by it. It can be shown that degree and tree-size for polynomial size Boolean circuits are polynomially related.

The *degree*, $D(v)$, of a vertex v in a Boolean circuit G_n with n inputs is defined as follows:

- If v is a vertex (of indegree 0) labeled from $\{0,1\}$, $D(v)=0$.
- If v is a vertex (of indegree 0) labeled from the set $\{x_1,x_2,\ldots,x_n,\overline{x}_1,\overline{x}_2,\ldots,\overline{x}_n\}$, $D(v)=1$.
- If v is a gate labeled OR with $k \geq 2$ inputs $v_1,\ldots,v_k$, $D(v)=max\{D(v_1),\ldots,D(v_k)\}$.
- If v is a gate labeled AND with $k \geq 2$ inputs $v_1 \ldots v_k$, $D(v)=\sum_{i=1}^{k} D(v_i)$.

The degree of a circuit is the degree of its output gate. The following theorem relates the degree, tree-size, and depth of a circuit.

Theorem 4 Let D, Z, and d be the degree, tree-size, and depth respectively of a Boolean circuit G. Then,

$Z \leq D\,d+1$.

Proof Let x be an input accepted by G. By hypothesis, there is an accepting subtree H of G of size at most Z. Let v be any vertex in H. The theorem then follows from the claim below, which can be proved easily by induction on the depth of v.

Claim: Let $Z(v)$ denote the number of vertices in the subtree of H rooted at v, $D(v)$ be the degree of v and $d(v)$ be the depth of v. Then,

$Z(v) \leq D(v)\ d(v) + 1.\Box$

Before we conclude this section, it is appropriate to mention a complete problem for LOGCFL that also has this polynomial proof size property. Consider the *Path systems* model introduced by Cook [4]. By defining polynomial tree-solvable path systems as those in which the tree-equivalent of some solution graph has polynomial size, Sudborough [17] has shown that the recognition problem for this class is complete for LOGCFL.

4 Semi-unbounded Fan-in Circuits

Perhaps the most interesting of the characterizations in this paper is the one using semi-unbounded fan-in circuits. Its claim to interest is due to the known characterization of AC^1 using unbounded fan-in circuits [20]. Thus, semi-unbounded fan-in circuits provide a combinatorial setting, namely the circuit model, in which the separation problem for the classes LOGCFL and AC^1 can be studied. This question in terms of these two models is whether the set of languages accepted by polynomial size semi-unbounded fan-in Boolean circuits within logarithmic depth is a proper subset of the languages accepted by polynomial size unbounded fan-in circuits within logarithmic depth.

Recently, Borodin et al. [2] showed that the class of languages accepted by polynomial size semi-unbounded fan-in circuits within logarithmic depth is closed under complement. Since language classes defined using the unbounded fan-in circuit model are closed under complement, this approach is ruled out in trying to separate the classes defined using these two models. But, it is interesting to observe that the constant depth analogs of these models exhibit this nice separation: constant depth unbounded fan-in circuits are closed under complement whereas constant depth semi-unbounded fan-in circuits are not closed under complement. This is demonstrated below.

Theorem 5 There is a language accepted by a constant depth semi-unbounded fan-in circuit whose complement is not accepted by any constant depth semi-unbounded fan-in circuit.

Proof Let $n = 2m$ for some integer $m > 0$. Consider the language L_n defined as the set of all length n bit strings such that the first m bits are not equal to the last m bits. The desired language is the union of the sets L_n for all even n. Then a constant depth semi-unbounded fan-in circuit for L_n is given by the characterization: $x = y_1 \cdots y_m z_1 \cdots z_m$ is in L_n iff there exists i with $1 \le i \le m$ such that $y_i \neq z_i$.

Consider the complement $\overline{L_n}$ of L_n, that is the set of all length n bit strings such that the first m bits is the same as the last m bits. Suppose there is a constant depth semi-unbounded fan-in circuit G_n that accepts $\overline{L_n}$. Then, by lemma 6 below, $\overline{L_n}$ should have at least 2^{n-c} strings for some constant $c > 0$. But, there are exactly $2^m = 2^{n/2}$ strings in $\overline{L_n}$. This is a contradiction and hence the theorem holds. □

Lemma 6 The number of inputs accepted by a constant depth semi-unbounded fan-in circuit G_n is either zero or at least 2^{n-c} for some constant $c > 0$.

Proof Let x be an input to G_n on which it evaluates to one. Therefore, by fact 1, there is an accepting subtree H of G_n on input x. Then the claim below can be proved easily by induction on d.

Claim: For any gate v at depth d in H, the number of input vertices included in the subtree rooted at v is at most k^d, where k is the maximum of the fan-in of all AND gates in H.

Let $x = x_1 \cdots x_n$. By the claim above, the inputs included in H get values from at most $c = k^d > 0$, a constant number, of the bits from $\{x_1, \cdots, x_n\}$. Let the positions of these bits be $i_1, \cdots, i_c$. Then G_n evaluates to one on all n bit inputs that have the same values at $i_1, \cdots, i_c$ as x. This proves the lemma. □

This proof actually shows something stronger, namely that the language $\overline{L_n}$ cannot be accepted by any semi-unbounded fan-in circuit of depth $o(\log n)$, even if semi-unboundedness were weakened to allow poly-log fan-in to the AND gates in the circuit.

5 Conclusion

In this paper, we have attempted to isolate the key properties that characterize the parallel complexity class LOGCFL. One of these properties, namely, semi-unboundedness, led us to define a new model of computation based on unbounded fan-in circuits. We believe this to be one of the main contributions of this work. We were also able to unify the many characterizations of LOGCFL using the notion of polynomial proof size inherent in these characterizations.

New characterizations of a complexity class can serve as tools for showing membership in the class. They also suggest new complete problems for the class. We hope the characterizations in this paper would also be useful in these respects.

Finally, we conclude with some interesting questions raised by this work.

The semi-unbounded fan-in circuit model seems to be an interesting model in its own right. We believe that it will lead to new insights into computation just as the study of unbounded fan-in circuits has proved to be a rewarding one. Some recent results characterizing important sequential complexity classes such as $\mathcal{NP}$ are presented in [22].

The circuit characterization of LOGCFL illustrates how fan-in of gates is an important resource in this model. Thus, for instance, a major difference between the three classes NC^1, LOGCFL and AC^1 is the fan-in of the gates in the circuits that define these classes. Here NC^1 is the class languages accepted by uniform families of circuits with polynomial size and $O(\log n)$ depth with the OR and AND gates having bounded fan-in [6].

The circuit characterization of LOGCFL also provides a combinatorial framework to study the properties of the class LOGCFL. Recently, Borodin et al. [2] have shown that the class LOGCFL is closed under complement using the semi-unbounded fan-in circuit characterization of LOGCFL presented in this paper.

It would be interesting to isolate the semi-unboundedness property for the Parallel Random Access Machine (PRAM) model to obtain characterizations of LOGCFL on these models. It is relevant to note in this connection that both AC^1 and LOGDCFL(the class of languages log space reducible to deterministic context-free languages) have been characterized using the PRAM model ([20], [8]).

Acknowledgements

I am greatly indebted to Larry Ruzzo and Martin Tompa for clarifying many of the underlying ideas. I am very grateful to Richard Ladner for his valuable comments. My thanks are also due to the two anonymous referees for their helpful suggestions.

References

[1] Borodin, A., On Relating Time and Space to Size and Depth, *SIAM Journal of Computing* **6**, (1977), 733-743.

[2] Borodin, A., S.A. Cook, P.W. Dymond, W.L. Ruzzo, and M. Tompa, Two Applications of Inductive Counting for Complementation Problems, *SIAM Journal of Computing* **18**, (1989), 559-578.

[3] Chandra, A.K., D.C. Kozen, and L.J. Stockmeyer, Alternation, *Journal of the Association for Computing Machinery* **28**, (1981), 114-133.

[4] Cook, S.A., Path Systems and Language Recognition, *Proc. 2nd Annual Symposium on Symposium on Theory of Computing* (1970), 70-72.

[5] Cook, S.A., Characterizations of Pushdown Machines in terms of Time-bounded Computers, *Journal of the Association for Computing Machinery* **18**, 1 (Jan 1971), 4-18.

[6] Cook, S.A., A Taxonomy of Problems with Fast Parallel Algorithms, *Information and Control* **64**, 1-3 (Jan/Feb/Mar 1985), 2-22.

[7] Dahlhaus, E. and M.K. Warmuth, Membership for Growing Context-Sensitive Grammars is Polynomial, *Journal of Computer and System Sciences* **33**, (1986), 456-472.

[8] Dymond, P.W. and W.L. Ruzzo, Parallel Random Access Machines with Owned Global Memory and Deterministic Context-Free Language Recognition, *in* "International Colloquium on Automata, Languages, and Programming", Lecture Notes in Computer Science, No. 226, Springer Verlag, (1986), 95-104.

[9] Engelfriet, J., The Complexity of Languages Generated by Attribute Grammars, *SIAM Journal of Computing* **15**, (Feb 1986), 70-86.

[10] Goldschlager, L.M., The Monotone and Planar Circuit Value Problems are log space Complete for P, *SIGACT News* **9**, 2, 1977, 25-29.

[11] Immerman, N., Upper and Lower Bounds for First Order Expressiblity, *Journal of Computer and System Sciences* **25**, (1982), 76-98.

[12] Ladner, R.E., R.J. Lipton, and L.J. Stockmeyer, Alternating Pushdown and Stack Automata, *SIAM Journal of Computing* **13**, (Feb 1984), 135-155.

[13] Lewis, P.M., R.E. Stearns, and J. Hartmanis, Memory Bounds for Recognition of Context-free and Context Sensitive Languages, *Proc. IEEE 6th Annual Symposium on Switching Theory and Logic Design*, 1965, 191-202.

[14] Ruzzo, W.L., Tree-Size Bounded Alternation, *Journal of Computer and System Sciences* **20**, (1980), 218-235.

[15] Ruzzo, W.L., On Uniform Circuit Complexity, *Journal of Computer and System Sciences* **22**, (1981), 365-383.

[16] Skyum, S., L.G. Valiant, A Complexity Theory Based on Boolean Algebra, *Journal of the Association for Computing Machinery* **32**, (1985), 484-502.

[17] Sudborough, I.H., Time and Tape Bounded Auxiliary Pushdown Automata, *in* "Mathematical Foundations of Computer Science", Lecture Notes in Computer Science, No. 53, Springer Verlag, (1977), 493-503.

[18] Sudborough, I.H., The Complexity of the Membership Problem for some Extensions of Context-Free Languages, *International Journal of Computer Mathematics A* **6**, (1977), 191-215.

[19] Sudborough, I.H., On the Tape Complexity of Deterministic Context-free languages, *Journal of the Association for Computing Machinery* **25**, 3(1978), 405-414.

[20] Stockmeyer, L. and U. Vishkin, Simulation of Parallel Random Access Machines by Circuits, *SIAM Journal of Computing* **13**, (1984), 409-422.

[21] Venkateswaran, H., Properties That Characterize LOGCFL, *Proc. 19th Annual ACM Symposium on Theory of Computing*, (1987), 141-150.

[22] Venkateswaran, H., Circuit Definitions of Nondeterministic Complexity Classes, *Proc. 8th Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, (1988), Pune, India.

[23] Venkateswaran, H. and M. Tompa, A New Pebble Game That Characterizes Parallel Complexity Classes, *SIAM Journal of Computing* **18**, (1989), 533-549.