Making Nondeterminism Unambiguous

Klaus Reinhardt*
Wilhelm-Schickard Institut für Informatik
Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
e-mail: reinhard@informatik.uni-tuebingen.de

Eric Allender†
Department of Computer Science, Rutgers University
P.O. Box 1179, Piscataway, NJ 08855-1179,USA
e-mail: allender@cs.rutgers.edu

April 28, 1997

Abstract

We show that in the context of nonuniform complexity, nondeterministic logarithmic space bounded computation can be made unambiguous. An analogous result holds for the class of problems reducible to context-free languages. In terms of complexity classes, this can be stated as:

$$\mathrm{NL/poly} = \mathrm{UL/poly}$$
$$\mathrm{LogCFL/poly} = \mathrm{UAuxPDA}(\log n, n^{O(1)})/\mathrm{poly}$$

1 Introduction

In this paper, we combine two very useful algorithmic techniques (the inductive counting technique of [Imm88, Sze88] and the isolation lemma of [MVV87]) to give a simple proof that two fundamental concepts in complexity theory coincide in the context of nonuniform computation.

Unambiguous computation has been the focus of much attention over the past three decades. Unambiguous context-free languages form one of the most important subclasses of the class of context-free languages. The complexity class UP was first defined and studied by Valiant [Val76]; a necessary precondition for the existence of one-way functions is for P to be properly contained in UP

*Supported in part by the DFG Project La 618/3-1 KOMET.
†Supported in part by NSF grant CCR-9509603. This work was performed while this author was a visiting scholar at the Wilhelm-Schickard Institut für Informatik, Universität Tübingen, supported by DFG grant TU 7/117-1

[GS88]. Although UP is one of the most intensely-studied subclasses of NP, it is neither known nor widely-believed that UP contains any sets that are hard for NP under any interesting notion of reducibility. (Although Valiant and Vazirani showed that "*Unique.Satisfiability*" is hard for NP under probabilistic reductions [VV86], the language *Unique.Satisfiability* is not in UP unless NP = coNP.)

Nondeterministic and unambiguous space-bounded computation have also been the focus of much work in computer science. Nondeterministic logspace (NL) captures the complexity of many natural computational problems. The proof that NL is closed under complementation [Imm88, Sze88] answered the long-standing open question of whether the complement of every context-sensitive language is context-sensitive. It remains an open question if every context-sensitive language has an unambiguous grammar. The unambiguous version of NL, denoted UL, was first explicitly defined and studied in [BJLR92, AJ93]. A language A is in UL if and only if there is a nondeterministic logspace machine M accepting A such that, for every x, M has at most one accepting computation on input x.

Motivated in part by the question of whether a space-bounded analog of the result of [VV86] could be proved, Wigderson [Wig94, GW96] proved the inclusion NL/poly $\subseteq \oplus$L/poly. This is a weaker statement than NL $\subseteq \oplus$L, which is still not known to hold. $\oplus$L is the class of languages A for which there is a nondeterministic logspace bounded machine M such that $x \in A$ if and only M has an odd number of accepting computation paths on input x. Given any complexity class $\mathcal{C}$, $\mathcal{C}$/poly is the class of languages A for which there exists a sequence of "advice strings" $\{\alpha(n) \mid n \in \mathbf{N}\}$ and a language $B \in \mathcal{C}$ such that $x \in A$ if and only if $(x, \alpha(|x|)) \in B$. Classes of the form $\mathcal{C}$ provide a simple link between (nonuniform) circuit complexity classes, and machine-based complexity classes (such as P, NP, NL, $\oplus$L, etc.) that have natural characterizations in terms of *uniform* circuit families.

(It is worth emphasizing that, in showing the equality UL/poly = NL/poly, we must show that for every B in NL/poly, there is a nondeterministic logspace machine M that never has more than one accepting path on any input, and there is an advice sequence $\alpha(n)$ such that $M(x, \alpha(|x|))$ accepts if and only $x \in B$. This is stronger than merely saying that there is an advice sequence $\alpha(n)$ and a nondeterministic logspace machine such that $M(x, \alpha(|x|))$ never has more than one accepting path, and it accepts if and only if $x \in B$.)

In the proof of the main result of [Wig94, GW96], Wigderson observed that a simple modification of his construction produces graphs in which the shortest distance between every pair of nodes is achieved by a unique path. We will refer to such graphs in the following as *min-unique graphs*. Wigderson wrote: "We see no application of this observation." The proof of our main result is just such an application.

2 Nondeterministic Logspace

The s-t connectivity problem takes as input a directed graph with two distinguished vertices s and t, and determines if there is a path in the graph from s to t. It is well-known that this is a complete problem for NL [Jon75].

The following lemma is implicit in [Wig94, GW96], but for completeness we make it explicit here.

Lemma 2.1 *There is a logspace-computable function f and a sequence of "advice strings" $\{\alpha(n) \mid n \in \mathbf{N}\}$ (where $|\alpha(n)|$ is bounded by a polynomial in n) with the following properties:*

- *For any graph G on n vertices, $f(G, \alpha(n)) = \langle G_1, \ldots, G_{n^2} \rangle$.*
- *For each i, the graph G_i has an s-t path if and only if G has an s-t path.*
- *If G has an s-t path then there is some i such that G_i is a min-unique graph.*

Proof: We first observe that a standard application of the isolation lemma technique of [MVV87] shows that, if each edge in G is assigned a weight in the range $[1, 4n^4]$ uniformly and independently at random, then with probability at least $\frac{3}{4}$, for any two vertices x and y such that there is a path from x to y, there is only one path having minimum weight. (Sketch: The probability that there is more than one minimum weight path from x to y is bounded by the sum, over all edges e, of the probability of the event $\text{BAD}(e, x, y)$ ::= "e occurs on one minimum-weight path from x to y and not in another". Given any weight assignment w' to the edges in G other than e, there is at most one value z with the property that, if the weight of e is set to be z, then $\text{BAD}(e, x, y)$ occurs. Thus the probability that there are two minimum-weight paths between two vertices is bounded by $\sum_{x,y,e} \sum_{w'} \text{BAD}(e, x, y|w')\text{Prob}(w')$ $\leq \sum_{x,y,e} \sum_{w'} 1/(4n^4)\text{Prob}(w') = \sum_{x,y,e} 1/(4n^4) \leq 1/4$.)

Our advice string α will consist of a sequence of n^2 weight functions, where each weight function assigns a weight in the range $[1, 4n^4]$ to each edge. (There are $A(n) = 2^{O(n^5)}$ such advice strings possible for each n.) Our logspace-computable function f takes as input a graph G and a sequence of n^2 weight functions, and produces as output a sequence of graphs $\langle G_1, \ldots, G_{n^2} \rangle$, where graph G_i is the result of replacing each edge $e = (x, y)$ in G by a path of length j from x to y, where j is the weight given to e by the i-th weight function in the advice string. Note that, if the i-th weight function satisfies the property that there is at most one minimum weight path between any two vertices, then G_i is a min-unique graph. (It suffices to observe that, for any two vertices x and y of G_i, there are vertices x' and y' such that

- x' and y' are vertices of the original graph G, and they lie on every path between x and y,
- there is only one path from x to x', and only one path from y' to y, and

- the minimum weight path from x to y is unique.)

Let us call an advice string "bad for G" if none of the graphs G_i in the sequence $f(G)$ is a min-unique graph. For each G, the probability that a randomly-chosen advice string α is bad is bounded by (probability that G_i is not min-unique)$^{n^2} \leq (1/4)^{n^2} = 2^{-2n^2}$. Thus the total number of advice strings that are bad for some G is at most $2^{n^2}(2^{-2n^2}A(n)) < A(n)$. Thus there is some advice string $\alpha(n)$ that is not bad. ■

Theorem 2.2 *NL⊆UL/poly*

Proof: It suffices to present a UL/poly algorithm for the s-t connectivity problem.

We show that there is a nondeterministic logspace machine M that takes as input a sequence of digraphs $\langle G_1, \ldots, G_r\rangle$, and processes each G_i in sequence, with the following properties:

- If G_i is not min-unique, M has a unique path that determines this fact and goes on to process G_{i+1};[1] all other paths are rejecting.
- If G_i is a min-unique graph with an s-t path, then M has a unique accepting path.
- If G_i is a min-unique graph with no s-t path, then M has no accepting path.

Combining this routine with the construction of Lemma 2.1 yields the desired UL/poly algorithm.

Our algorithm is an enhancement of the inductive counting technique of [Imm88] and [Sze88]. We call this the *double counting* technique since in each stage we count not only the number of vertices having distance at most k from the start vertex, but also the sum of the lengths of the shortest path to each such vertex. In the following description of the algorithm, we denote these numbers by c_k and Σ_k, respectively.

Let us use the notation $d(v)$ to denote the length of the shortest path in a graph G from the start vertex to v. (If no such path exists, then $d(v) = n + 1$.) Thus, using this notation, $\Sigma_k = \sum_{\{x|d(x)\leq k\}} d(x)$.

A useful observation is that *if the subgraph of G having distance at most k from the start vertex is min-unique* (and if the correct values of c_k and Σ_k are provided), then an unambiguous logspace machine can, on input (G, k, c_k, Σ_k, v), compute the Boolean predicate "$d(v) \leq k$". This is achieved with the routine shown in Figure 1.

To see that this routine truly is unambiguous if the preconditions are met, note the following:

[1] More precisely, our routine will check if, for every vertex x, there is at most one minimal-length path from the start vertex to x. This is sufficient for our purposes. A straightforward modification of our routine would provide an unambiguous logspace routine that will determine if the entire graph G_i is a min-unique graph.

```
Input (G, k, c_k, Σ_k, v)
count := 0; sum := 0; path.to.v := false;
for each x ∈ V do
    Guess nondeterministically if d(x) ≤ k.
    if the guess is d(x) ≤ k then
        begin
         Guess a path of length l ≤ k from s to x
             (If this fails, then halt and reject).
         count := count +1; sum := sum +l;
         if x = v then path.to.v := true;
        end
endfor
if count = c_k and sum = Σ_k
    then return the Boolean value of path.to.v
    else halt and reject
end.procedure
```

Figure 1: An unambiguous routine to determine if $d(v) \leq k$.

- If the routine ever guesses incorrectly for some vertex x that $d(x) > k$, then the variable *count* will never reach c_k and the routine will reject. Thus the only paths that run to completion guess correctly exactly the set $\{x \mid d(x) \leq k\}$.
- If the routine ever guesses incorrectly the length l of the shortest path to x, then if $d(x) > l$ no path of length l will be found, and if $d(x) < l$ then the variable *sum* will be incremented by a value greater than $d(x)$. In the latter case, at the end of the routine, *sum* will be greater than Σ_k, and the routine will reject.

Clearly, the subgraph having distance at most 0 from the start vertex is min-unique, and $c_0 = 1$ and $\Sigma_0 = 0$. A key part of the construction involves computing c_k and Σ_k from c_{k-1} and Σ_{k-1}, at the same time checking that the subgraph having distance at most k from the start vertex is min-unique. It is easy to see that c_k is equal to c_{k-1} plus the number of vertices having $d(v) = k$. Note that $d(v) = k$ if and only if it is not the case that $d(v) \leq k - 1$ and there is some edge (x, v) such that $d(x) \leq k - 1$. The graph fails to be a min-unique graph if and only if there exist some v and x as above, as well as some other $x' \neq x$ such that $d(x') \leq k$ and there is an edge (x', v). The code shown in Figure 2 formalizes these considerations.

Searching for an s-t path in graph G is now expressed by the routine shown in Figure 3.

Given the sequence $\langle G_1, \ldots, G_r \rangle$, the routine processes each G_i in turn. If G_i is not min-unique (or more precisely, if the subgraph of G_i that is reachable from the start vertex is not a min-unique graph), then one unique computation path of the routine returns the value *BAD.GRAPH* and goes on to process G_{i+1};

Input $(G, k, c_{k-1}, \Sigma_{k-1})$
Output (c_k, Σ_k), and also the flag *BAD.GRAPH*

$c_k := c_{k-1}$; $\Sigma_k := \Sigma_{k-1}$;
for each vertex v **do**
    if $\neg(d(v) \leq k-1)$ **then**
        for each x such that (x, v) is an edge **do**
            if $d(x) \leq k-1$ **then**
                begin
                $c_k := c_k + 1; \Sigma_k := \Sigma_k + k;$
                for $x' \neq x$ **do**
                    if (x', v) is an edge **and** $d(x') \leq k-1$
                        then $BAD.GRAPH :=$ true:
                endfor
                end
        endfor
endfor At this point, the values of c_k and Σ_k are correct.

Figure 2: Computing c_k and Σ_k.

Input (G)
$BAD.GRAPH :=$ false; $c_0 := 1; \Sigma_0 := 0; k := 0;$
repeat
    $k := k + 1$:
    compute c_k and Σ_k from (c_{k-1}, Σ_{k-1});
until $c_{k-1} = c_k$ **or** $BAD.GRAPH =$ true.
If $BAD.GRAPH =$ false then there is an s-t path in G if and only if $d(t) \leq k$.

Figure 3: Finding an s-t path in a min-unique graph.

all other computation paths halt and reject. Otherwise, if G_i is min-unique, the routine has a unique accepting path if G_i has an s-t path, and if this is not the case the routine halts with no accepting computation paths. ■

Corollary 2.3 *NL/poly = UL/poly*

3 LogCFL

LogCFL is the class of problems logspace-reducible to a context-free language. Two important and useful characterizations of this class are summarized in the following proposition. (SAC^1 and AuxPDA$(\log n, n^{O(1)})$ are defined in the following paragraphs.)

Proposition 3.1 *LogCFL = AuxPDA*$(\log n, n^{O(1)})$ = *SAC*^{1} *[Sud78, Ven91]*

An Auxiliary Pushdown Automaton (AuxPDA) is a nondeterministic Turing machine with a read-only input tape, a space-bounded worktape, and a pushdown store that is not subject to the space-bound. The class of languages accepted by Auxiliary Pushdown Automata in space $s(n)$ and time $t(n)$ is denoted by AuxPDA$(s(n), t(n))$. If an AuxPDA satisfies the property that, on every input x, there is at most one accepting computation, then the AuxPDA is said to be *unambiguous.* This gives rise to the class UAuxPDA$(s(n), t(n))$.

SAC^1 is the class of languages accepted by logspace-uniform semi-unbounded circuits of depth $O(\log n)$; a circuit family is semi-unbounded if the AND gates have fan-in 2 and the OR gates have unbounded fan-in.

Not long after NL was shown to be closed under complementation [Imm88, Sze88], LogCFL was also shown to be closed under complementation in a proof that also used the inductive counting technique ([BCD$^+$89]). A similar history followed a few years later: not long after it was shown that NL is contained in $\oplus$L/poly [Wig94, GW96], the isolation lemma was again used to show that LogCFL is contained in $\oplus$SAC^1/poly [Gál95, GW96]. (As is noted in [GW96], this was independently shown by H. Venkateswaran.)

In this section, we show that the same techniques that were used in Section 2 can be used to prove an analogous result about LogCFL. (In fact, it would also be possible to derive the result of Section 2 from a modification of the proof of this section. Since some readers may be more interested in NL than LogCFL, we have chosen to present a direct proof of NL/poly = UL/poly.) The first step is to state the analog to Lemma 2.1. Before we can do that, we need some definitions.

A *weighted circuit* is a semiunbounded circuit together with a *weighting function* that assigns a nonnegative integer weight to each wire connecting any two gates in the circuit.

Let C be a weighted circuit, and let g be a gate of C. A *certificate for* $g(x) = 1$ *(in C)* is a list of gates, corresponding to a depth-first search of the subcircuit of C rooted at g. The *weight of a certificate* is the sum of the weights of the edges traversed in the depth-first search. This informal definition is made precise by the following inductive definition. (It should be noted that this definition differs in some unimportant ways from the definition given in [Gál95, GW96].)

- If g is a constant 1 gate or an input gate evaluating to 1 on input x, then the only certificate for g is the string g. This certificate has weight 0.
- If g is an AND gate of C with inputs h_1 and h_2 (where h_1 lexicographically precedes h_2), then any string of the form gyz is a certificate for g, where y is any certificate for h_1, and z is any certificate for h_2. If w_i is the weight of the edge connecting h_i to g, then the weight of the certificate yz is $w_1 + w_2$ plus the sum of the weights of certificates y and z.
- If g is an OR gate of C, then any string of the form gy is a certificate for g, where y is any certificate for a gate h that is an input to g in C. If w is the weight of the edge connecting h to g, then the weight of the certificate gy is w plus the weight of certificate y.

Note that if C has logarithmic depth d, then any certificate has length bounded by a polynomial in n and has weight bounded by 2^d times the maximum weight of any edge. Every gate that evaluates to 1 on input x has a certificate, and no gate that evaluates to 0 has a certificate.

We will say that a weighted circuit C *is min-unique on input* x if, for every gate g that evaluates to 1 on input x, the minimal-weight certificate for $g(x) = 1$ is unique.

Lemma 3.2 *For any language A in LogCFL, there is a sequence of advice strings $\alpha(n)$ (having length polynomial in n) with the following properties:*

- *Each $\alpha(n)$ is a list of weighted circuits of logarithmic depth $\langle C_1, \ldots, C_r \rangle$.*
- *For each input x and for each i, $x \in A$ if and only if $C_i(x) = 1$.*
- *For each input x, if $x \in A$, then there is some i such that C_i is min-unique on input x.*

Lemma 3.2 is in some sense implicit in [Gál95, GW96]. We include a proof for completeness.

Proof: Let A be in LogCFL, and let C be the semiunbounded circuit of size n^k and depth $d = O(\log n)$ recognizing A on inputs of length n.

As in [Gál95, GW96], a modified application of the isolation lemma technique of [MVV87] shows that, for each input x, if each wire in C is assigned a weight in the range $[1, 4n^{3k}]$ uniformly and independently at random, then with probability at least $\frac{3}{4}$, C is min-unique on input x. (Sketch: The probability that there is more than one minimum weight certificate for $g(x) = 1$ is bounded by the sum, over all wires e, of the probability of the event $\textsc{Bad}(e, g)$ ::= "e occurs in one minimum-weight certificate for $g(x) = 1$ and not in another". Given any weight assignment w' to the edges in C other than e, there is at most one value z with the property that, if the weight of e is set to be z, then

$\textsc{Bad}(e, g)$ occurs. Thus the probability that there are two minimum-weight certificates for any gate in C is bounded by $\sum_{g,e} \sum_{w'} \textsc{Bad}(e, g|w')\text{Prob}(w') \leq \sum_{g,e} \sum_{w'} 1/(4n^{3k})\text{Prob}(w') = \sum_{g,e} 1/(4n^{3k}) \leq 1/4.$)

Now consider sequences β consisting of n weight functions $\langle w_1, \ldots, w_n \rangle$, where each weight function assigns a weight in the range $[1, 4n^{3k}]$ to each edge of C. (There are $B(n) = 2^{n^{O(1)}}$ such sequences possible for each n.) There must exist a string β such that, for each input x of length n, there is some $i \leq n$ such that the weighted circuit C_i that results by applying weight function w_i to C is min-unique on input x. (*Sketch of proof:* Let us call a sequence β "bad for x" if none of the circuits C_i in the sequence is min-unique on input x. For each x, the probability that a randomly-chosen β is bad is bounded by (probability that C_i is not min-unique)$^n \leq (1/4)^n = 2^{-2n}$. Thus the total number of sequences that are bad for some x is at most $2^n(2^{-2n^2}B(n)) < B(n)$. Thus there is some sequence β that is not bad.)

The desired advice sequence $\alpha(n) = \langle C_1, \ldots, C_r \rangle$ is formed by taking a good sequence $\beta = \langle w_1, \ldots, w_n \rangle$ and letting C_i be the result of applying weight function w_i to C. ■

Theorem 3.3 *LogCFL* $\subseteq$ *UAuxPDA*$(\log n, n^{O(1)})$*/poly.*

Proof: Let A be a language in LogCFL. Let x be a string of length n, and let $\langle C_1, \ldots, C_r \rangle$ be the advice sequence guaranteed by Lemma 3.2.

We show that there is an unambiguous auxiliary pushdown automaton M that runs in polynomial time and uses logarithmic space on its worktape that, given a sequence of circuits as input, processes each circuit in turn, and has the following properties:

- If C_i is not min-unique on input x, then M has a unique path that determines this fact and goes on to process C_{i+1}; all other paths are rejecting.
- If C_i is min-unique on input x and evaluates to 1 on input x, then M has a unique accepting path.
- If C_i is min-unique on input x but evaluates to zero on input x, then M has no accepting path.

Our construction is similar in many respects to that of Section 2. Given a circuit C, let c_k denote the number of gates g that have a certificate for $g(x) = 1$ of weight at most k, and let Σ_k be the sum, over all gates g having a certificate for $g(x) = 1$ of weight at most k, of the minimum-weight certificate of g. (Let $W(g)$ denote the weight of the minimum-weight certificate of $g(x) = 1$, if such a certificate exists, and let this value be ∞ otherwise.)

A useful observation is that *if all gates of C having certificates of weight at most k have unique minimal-weight certificates* (and if the correct values of c_k and Σ_k are provided), then an unambiguous AuxPDA can, on input $(C, x, k, c_k, \Sigma_k, g, a)$, compute the Boolean value of the predicate "$W(g) = a \leq k$". This is achieved with the routine shown in Figure 4.

```
Input (C, x, k, c_k, Σ_k, g)
count := 0; sum := 0; a := ∞;
for each gate h do
      Guess nondeterministically if W(h) ≤ k.
      if the guess is W(h) ≤ k then
           begin
            Guess a certificate of size l ≤ k for h
                  (If this fails, then halt and reject).
            count := count +1; sum := sum +l;
            if h = g then a := l;
           end
endfor
if count = c_k and sum = Σ_k
      then return a
      else halt and reject
end.procedure
```

Figure 4: An unambiguous routine to calculate $W(g)$ if $W(g) \leq k$ and return ∞ otherwise.

To see that this routine truly is unambiguous if the preconditions are met, note the following:

- If the routine ever guesses incorrectly for some gate h that $W(h) > k$, then the variable *count* will never reach c_k and the routine will reject. Thus the only paths that run to completion guess correctly exactly the set $\{h \mid W(h) \leq k\}$.
- For each gate h such that $W(h) \leq k$, there is exactly one minimal-weight certificate that can be found. An UAuxPDA will find this certificate using its pushdown to execute a depth-first search (using nondeterminism at the OR gates, and using its $O(\log n)$ workspace to compute the weight of the certificate), and only one path will find the the minimal-weight certificate. If, for some gate h, a certificate of weight greater than $W(h)$ is guessed, then the variable *sum* will not be equal to Σ_k at the end of the routine, and the path will halt and reject.

Clearly, all gates at the input level have unique minimal-weight certificates (and the only gates g with $W(g) = 0$ are at the input level). Thus we can set $c_0 = n + 1$ (since each input bit and its negation are provided, along with the constant 1), and $\Sigma_k = 0$. A key part of the construction involves computing c_k and Σ_k from (c_{k-1}, Σ_{k-1}), at the same time checking that no gate has two minimal-weight certificates of weight k. Consider each gate g in turn. If g is an AND gate with inputs h_1 and h_2 and weights w_1 and w_2 connecting g to these inputs, then $W(g) \leq k$ if and only if $(W(g) = l \leq k-1)$ or $((W(g) > k-1)$ and $(W(h_1)+W(h_2)+w_1+w_2 = k))$. If g is an OR gate, then it suffices to check, for each gate h that is connected to g by an edge of weight w, if $(W(g) = l \leq k-1)$

or $((W(g) > k-1)$ and $(W(h) + w = k))$; if one such gate is found, then $W(g) = k$; if two such gates are found, then the circuit is not min-unique on input x. If no violations of this sort are found for any k, then C is min-unique on input x. The code shown in Figure 5 formalizes these considerations.

Input $(C, x, k, c_{k-1}, \Sigma_{k-1})$
Output (c_k, Σ_k), and also the flag *BAD.CIRCUIT*

```
c_k := c_{k-1}; Σ_k := Σ_{k-1};
for each gate g do
    if W(g) > k − 1 then
        begin
        if g is an AND gate with inputs h_1, h_2, connected to g
                with edges weighted w_1, w_2 and
                W(h_1) + W(h_2) + w_1 + w_2 = k then
            c_k := c_k + 1; Σ_k := Σ_k + k
        if g is an OR gate then
            for each h connected to g by an edge weighted w do
                if W(h) = k − w then
                    begin
                    c_k := c_k + 1; Σ_k := Σ_k + k
                    for h' ≠ h connected to g by an edge of weight w' do
                        if W(h') = k − w'
                            then BAD.CIRCUIT := true:
                    endfor
                    end
            endfor
        end
endfor
```

At this point, if *BAD.CIRCUIT* = false, the values of c_k and Σ_k are correct.

Figure 5: Computing c_k and Σ_k.

Evaluating a given circuit C_i is now expressed by the routine shown in Figure 6.

Given the sequence $\langle C_1, \ldots, C_r\rangle$, the routine processes each C_i in turn. If C_i is not is min-unique on input x, then one unique computation path of the routine returns the value *BAD.CIRCUIT* and goes on to process C_{i+1}; all other computation paths halt and reject. Otherwise, the routine has a unique accepting path if $C_i(x) = 1$, and if this is not the case the routine halts with no accepting computation paths. ■

Corollary 3.4 *LogCFL/poly* $=$ *UAuxPDA*$(\log n, n^{O(1)})$*/poly.*

Input (C_i)
$\mathit{BAD.CIRCUIT}$:= false; $c_0 := n+1; \Sigma_0 := 0$;
for $k = 1$ **to** $2^d 4n^3$
    compute (c_k, Σ_k) from c_{k-1}, Σ_{k-1};
    if $\mathit{BAD.CIRCUIT}$ = true, then exit the **for** loop.
endfor
If $\mathit{BAD.CIRCUIT}$ = false then if the output gate g evaluates to 1, then it has a unique minimal-weight certificate of some weight l.
Accept if and only if $W(g) \neq \infty$

Figure 6: Evaluating a circuit.

4 Discussion and Open Problems

Rytter [Ryt87] (see also [RR92]) showed that any unambiguous context-free language can be recognized in logarithmic time by CREW-PRAM. In contrast, no such CREW algorithm is known for any problem complete for NL, even in the nonuniform setting. The problem is that, although NL is the class of languages reducible to linear context-free languages, and although the class of languages accepted by deterministic AuxPDAs in logarithmic space and polynomial time coincides with the class of languages logspace-reducible to deterministic context-free languages, and LogCFL coincides with AuxPDA$(\log n, n^{O(1)})$, it is *not* known that UAuxPDA$(\log n, n^{O(1)})$ or UL is reducible to unambiguous context-free languages. The work of Niedermeier and Rossmanith does an excellent job of explaining the subtleties and difficulties here [NR95]. CREW algorithms are closely associated with a version of unambiguity called *strong unambiguity*. In terms of Turing-machine based computation, strong unambiguity means that, not only is there at most one path from the start vertex to the accepting configuration, but in fact there is at most one path between *any two configurations of the machine*.

Strongly unambiguous algorithms have more efficient algorithms than are known for general NL or UL problems. It is shown in [AL96] that problems in Strongly unambiguous logspace have deterministic algorithms using less than $\log^2 n$ space.

The reader is encouraged to note that, in a min-unique graph, the shortest path between *any two vertices* is unique. This bears a superficial resemblance to the property of strong unambiguity. We see no application of this observation.

It is natural to ask if the randomized aspect of the construction can be eliminated using some sort of derandomization technique to obtain the equality UL = NL.

A corollary of our work is that UL/poly is closed under complement. It remains an open question if UL is closed under complement, although some of the unambiguous logspace classes that can be defined using strong unambiguity are known to be closed under complement [BJLR92].

It is disappointing that the techniques used in this paper do not seem to provide any new information about complexity classes such as NSPACE(n) and NSPACE(2^n). It is straightforward to show that NSPACE($s(n)$) is contained in USPACE($s(n)$)/$2^{O(s(n))}$, but this is interesting only for sublinear $s(n)$.

There is a natural class of functions associated with NL, denoted FNL [AJ93]. This can be defined in several equivalent ways, such as

- The class of functions computable by NC^1 circuits with oracle gates for problems in NL.
- The class of functions f such that $\{(x, i, b) \mid \text{the } i\text{-th bit of } f(x) \text{ is } b\}$ is in NL.
- The class of functions computable by logspace-bounded machines with oracles for NL.

Another important class of problems related to NL is the class #L, which counts the number of accepting paths of a NL machine. #L characterizes the complexity of computing the determinant [Vin91]. (See also [Tod, Dam, MV97, Val92, AO96].) It was observed in [AJ93] that if NL = UL, then FNL is contained in #L. Thus a corollary of the result in this paper is that FNL/poly $\subseteq$ #L/poly.

Many questions about #L remain unanswered. Two interesting complexity classes related to #L are PL (probabilistic logspace) and $\mathrm{C_=L}$ (which characterizes the complexity of singular matrices, as well as questions about computing the rank). It is known that some natural hierarchies defined using these complexity classes collapse:

- $\mathrm{AC}^0(\mathrm{C_=L}) = \mathrm{C_=L}^{\mathrm{C_=L}^{\cdot^{\cdot^{\mathrm{C_=L}}}}} = \mathrm{NC}^1(\mathrm{C_=L}) = \mathrm{L}^{\mathrm{C_=L}}$ [AO96, ABO96].
- $\mathrm{AC}^0(\mathrm{PL}) = \mathrm{PL}^{\mathrm{PL}^{\cdot^{\cdot^{\mathrm{PL}}}}} = \mathrm{NC}^1(\mathrm{PL}) = \mathrm{PL}$ [AO96, Ogi96, BF].

In contrast, the corresponding #L hierarchy (equal to the class of problems AC^0 reducible to computing the determinant) $\mathrm{AC}^0(\#\mathrm{L}) = \mathrm{FL}^{\#\mathrm{L}^{\cdot^{\cdot^{\#\mathrm{L}}}}}$ is not known to collapse to any fixed level. Does the equality UL/poly = NL/poly provide any help in analyzing this hierarchy in the nonuniform setting?

Acknowledgment: We thank Klaus-Jörn Lange for helpful comments, and for drawing our attention to min-unique graphs, and for arranging for the second author to spend some of his sabbatical time in Tübingen. We also thank V. Vinay and Lance Fortnow for insightful comments.

References

[ABO96] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. In *ACM Symposium on Theory of Computing (STOC)*, 1996.

[AJ93] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

[AL96] E. Allender and K.-J. Lange. StUSPACE($\log n$) is contained in DSPACE($log^2 n/loglogn$). In *Proceedings of the 7th ACM-SIGSAM International Symposium on Symbolic and Algebraic Computation (ISAAC)*, volume 1178 of *Lecture Notes in Computer Science*, pages 193–202. Springer-Verlag, 1996.

[AO96] E. Allender and M. Ogihara. Relationships among PL, #L and the determinant. *RAIRO - Theoretical Information and Application*, 30:1–21, 1996.

[BCD+89] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal on Computing*, 18(3):559–578, 1989.

[BF] R. Beigel and B. Fu. Circuits over PP and PL. To appear in *Proceedings of the 12th Conference on Computational Complexity*, 1997.

[BJLR92] G. Buntrock, B. Jenner, K.-J. Lange, and P. Rossmanith. Unambiguity and fewness for logarithmic space. In *Proc. 8th International Conference on Fundamentals of Computation Theory (FCT '91)*, volume 529 of *Lecture Notes in Computer Science*, pages 168–179. Springer-Verlag, 1992.

[Dam] C. Damm. DET = $\mathrm{L}^{\#\mathrm{L}}$? Informatik-Preprint 8, Fachbereich Informatik der Humboldt-Universität zu Berlin, 1991.

[Gál95] A. Gál. Semi-unbounded fan-in circuits: Boolean vs. arithmetic. In *IEEE Structure in Complexity Theory Conference*, pages 82–87, 1995.

[GS88] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17:309–335, 1988.

[GW96] A. Gál and A. Wigderson. Boolean vs. arithmetic complexity classes: randomized reductions. *Random Structures and Algorithms*, 9:99–111, 1996.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988.

[Jon75] N. D. Jones. Space bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–85, 1975.

[MV97] M. Mahajan and V. Vinay. A combinatorial algorithm for the determinant. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1997.

[MVV87] K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

[NR95] R. Niedermeier and P. Rossmanith. Unambiguous auxiliary push-down automata and semi-unbounded fan-in circuits. *Information and Computation*, 118(2):227–245, 1995.

[Ogi96] M. Ogihara. The PL hierarchy collapses. In *ACM Symposium on Theory of Computing (STOC)*, pages 84–88, 1996.

[RR92] P. Rossmanith and W. Rytter. Observations on $\log n$ time parallel recognition of unambiguous context-free languages. *Information Processing Letters*, 44:267–272, 1992.

[Ryt87] W. Rytter. Parallel time $O(\log n)$ recognition of unambiguous context-free languages. *Information and Computation*, 73:75–86, 1987.

[Sud78] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *J. Association of Computing Machinery*, 25:405–414, 1978.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[Tod] S. Toda. Counting problems computationally equivalent to the determinant. Technical Report CSIM 91-07, Department of Computer Science and Information Mathematics, University of Electro-Communications, Tokyo, 1991.

[Val76] L. Valiant. The relative complexity of checking and evaluating. *Information Processing Letters*, 5:20–23, 1976.

[Val92] L. Valiant. Why is Boolean complexity theory difficult? In M. Paterson, editor, *Boolean Function Complexity*, volume 169 of *London Mathematical Society Lecture Notes Series*, pages 84–94. Cambridge University Press, 1992.

[Ven91] Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43, 1991.

[Vin91] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proc. 6th Structure in Complexity Theory Conference*, pages 270–284. IEEE, 1991.

[VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[Wig94] A. Wigderson. NL/poly $\subseteq \bigoplus$L/poly. In *Proc. of the 9th IEEE Structure in Complexity Conference*, pages 59–62, 1994.