Properties of $OptL$, $\#L$ and UL and their relation with NL class

A Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by
Koushik Sen (95131)
Vivek Tandon (95338)

to the

Department of Computer Science & Engineering

Indian Institute of Technology Kanpur

October, 2003

Certificate

Certified that the work contained in the report entitled "*Properties of OptL, #L and UL and their relation with NL class*", by Koushik Sen (95131) and Vivek Tandon (95338), has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Manindra Agarwal

October, 2003

Preface

We show that the *Opt* function of an NL machine having a polynomial number of distinct outputs can be computed by an $FL(NL)$ function. A similar result for $\#L$ class is also proved.Using similar technique we also prove that a restricted $FewL$ is contained in UL. It is also shown that, provided $UL =$ co-UL, the computation of an NL machine having a constant number of accepting outputs can be done unambiguously.

Acknowledgements

We would like to thank Dr. Manindra Agarwal for his invaluable guidance and immense patience, without whose help, our proofs would have never seen the light of the day.

Contents

Chapter 1

Introduction

Nondeterministic and unambiguous logspace-bounded computation have been the focus of much work in theoretical study in computer science. Of special interest in this category are the NL and the UL classes.

The $OptL$ class is an interesting counting variant of the class NL. The complexity of the $OptL$ class poses interesting questions, of particular interest is its relationship with the $FL(NL)$ class. Alvarez and Jenner [AJ93], have shown some interesting results regarding the $OptL$ class. They have shown that $FL(NL) \subseteq OptL$. An important result "$OptL_p \subseteq FL(NL)$", is proved here. For proving the above result, we use an interesting technique that allows us to distinguish the accepting paths for $FewL$ machine.

The technique is discussed in the following chapters. It is applied for making a nondeterministic class unambiguous.

It is an open question whether NL and the unambiguous version of NL, i.e, (UL), belong to the same complexity class. Reinhardt and Allender have shown that $NL \subseteq UL$/poly [RA97] using randomization and the concept of *min-unique* graphs. The technique developed in the earlier problem is used along with the concept of min-uniqueness to show that $UL = StrongFewL$.

It is not known whether $UL =$ co-UL. Assuming that this is true, it is shown here that $UL = NL$ with the restriction that there are at most a constant number of accepting paths, i.e, $UL = ConstL$.

The paper is organized as follows:

- **Chapter 2:** This chapter consists of the definitions and the results already known and relevant to this work.
- **Chapter 3:** This chapter talks about an important technique to distinguish between the accepting paths of $FewL$ machine and the relation between $OptL$ class and the $FL(NL)$ class for a restricted case is proved.
- **Chapter 4:** This chapter gives unambiguous machines for NL machines under certain restrictions.
- **Chapter 5 :** This chapter gives suggestions for future work.

The appendix consists of detailed proofs.

Chapter 2

Definitions and known results

2.1 Definitions

NL-transducer :A non-deterministic logarithmic space-bounded Turing machine with unbounded output tape and accepting and rejecting states. The output y of a computation of a transducer T on input x is the contents of $T's$ output tape when T halts. An output is considered to be *"valid"* if T halts in an accepting state.

FL(NL) : $\{f \mid f$ is computed by some L-transducer that has access to an oracle from NL $\}$

FNL : $\{f \mid f$ is computed by some single-valued NL-transducer $\}$

$\mathbf{f_{maxDFA}}, \mathbf{f_{maxNFA}}$: *Input* : An encoding of a DFA(NFA) M and a string $x \in \{0,1\}^*$.
Output : Lexicographically greatest word $y \in L(M)$ with $y \leq x$. (If no such word exists, output is $\perp$.)

$\mathbf{Opt_M}$: For a transducer M, opt_M denotes the function from $\{0,1\}^*$ to N such that $opt_M(x)$ is the maximum valid output value of M on input x with respect to lexicographic order.

OptL : $\{f \mid f = opt_T$ for some NL transducer T $\}$

$\mathbf{OptL_p}$: $\{f \mid f = opt_T$ for some NL transducer T having at most polynomial number of valid outputs$\}$.

$\mathbf{acc_M}$: For an NL machine M, acc_M denotes the function from $\{0,1\}^*$ to N such that $acc_M(x)$ is the number of accepting computations of M on x.

#L : $\{f \mid f = acc_M$ for some NL machine M $\}$

$\#\mathbf{L_p}$: $\{f \mid f = acc_M$ for some NL machine M having at most polynomial number of accepting paths $\}$.

UL : 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.

FewL : $FewL$ is the class of languages accepted by an NL machine having polynomial number of accepting paths.

C**/poly** : Given any complexity class C, 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 $B \in C$ such that $x \in A$ if and only if $(x, \alpha(\mid x \mid)) \in B$.

StrongFewL : $StrongFewL$ is the class of languages accepted by an NL machine with the restriction that there are at most polynomial number of paths between the starting configuration and any other configuration.

ConstL : $ConstL$ is the class of languages accepted by an NL machine having a constant number of accepting paths.

min-unique graphs : Graphs where the shortest distance between every pair of nodes is achieved by a *unique* path.

Critical Node: A node v of a graph $\mathbf{G}$[1] (directed graph without any cycle and having two distinct nodes s and t) is said to be a **critical node** if it is either s or t, or if v lies on a path from s to t and if either

- v is connected to s and forks into two new paths leading to t, or
- v is connected to t and two paths from s meet at v.

Critical subgraph: A **critical subgraph G'** of a graph $\mathbf{G}$[2] (directed graph without any cycle and having two distinct nodes s and t) is a subgraph of **G** consisting of only the **critical nodes** of **G** and satisfying the following properties:

- There is an edge from node $v1$ to $v2$ of **G'** if and only if there is a path from $v1$ to $v2$ in the graph **G** and no other critical node v lies on the path.
- If there are two paths from $v1$ to $v2$ in **G** containing no critical nodes, then there are two edges from $v1$ to $v2$ in graph **G'**.

It is obvious from the definition that the number of paths from s to t in both the graphs **G** and **G'** are same.

2.2 Known Results

Alvarej and Jenner [AJ93] have studied the properties of the $OptL$ and $FL(NL)$ classes and have obtained important results related to these two classes. The question of how to make nondeterminism unambiguous has been studied by Allender and Reinhardt [RA97]. Important results relevant to our work have been given below:

- It has been shown that f_{maxNFA} is *log-space many-one complete* for $OptL$ while $f_{maxDFA} \in FL(NL)$. [AJ93]

[1] the out degree of any node is assumed to be at most two
[2] the out degree of any node is assumed to be at most two

- $FL(NL) = FNL$. [AJ93]
- $FNL \subseteq OptL$. [AJ93]
- $NL = UL/\text{poly}$. [RA97]

Chapter 3

Relation between FL(NL) and OptL

Lemma 3.1 *Given numbers,* $k_1, k_2, \ldots, k_t$ *, where* $\forall i\ k_i \leq 2^n$ *and* $t \leq p(n)$ *and* $k_i \neq k_j$ *for all* $i \neq j$ *, there exists a prime number* $p < p^3(n)n^2$ *such that for all* i *and* j *and* $i \neq j$ *,* $k_i \bmod p \neq k_j \bmod p$*.*

Proof: Consider all primes p_1, ..., p_l upto $p^3(n)n^2$. By *Prime number theorem*, there are at least $\frac{p^3(n)n^2}{3\log p(n)+2\log n} > p^2(n)n$ primes. Suppose, for some fixed i and j, $i \neq j$, there are more than n primes such that for each such prime p, $k_i \equiv k_j$ (mod$p$). Then by *Chinese Remaindering Theorem* $k_i \equiv k_j$ modulo the product of all such primes. But the product of all such primes is greater than 2^n. Since both k_i and k_j are less than 2^n, we get $k_i = k_j$ which contradicts our assumption. Hence, there can be at most n such primes. Let us call these primes as **bad** primes. Now, for each pair k_i, k_j we have at most n bad primes. There are at most $p^2(n)$ pairs, hence, at most $p^2(n)n$ **bad** primes. Hence, we can find a prime p in the range 2 - $p^3(n)n^2$ such that for all i and j and $i \neq j$, $k_i \bmod p \neq k_j \bmod p$. ∎

Theorem 3.2 $OptL_p \subseteq FL(NL)$.

Proof: Let M be an NL transducer having polynomial number of valid outputs and let f be the Opt_M function. We show the construction of a $FL(NL)$ machine which compute the same function f.

The L-transducer first calculates l, the length of the maximum output of machine M on input x. For this, we place repeated queries to an oracle A^c, where A is defined as:

$$A = \{< M, x, i >| \text{ machine M has an output of length} \geq i \text{ on input } x\}.$$

Clearly, $A \subseteq NL$ and so is A^c. To calculate l, the L-transducer places repeated queries $(< M, x, 1 >, < M, x, 2 >, \ldots, < M, x, i >, \ldots)$ to the oracle A^c and finds the first i for which $< M, x, i >$ is in A^c. Hence, $l = i - 1$ and it can easily be seen that l is polynomial in n where $n =| x |$.

Machine M has polynomial number of valid outputs, each having a length bounded by a polynomial in n (say m). So, each output can be represented by m bits. By Lemma 3.1 it can be shown that there exists a prime $q \leq m^4$ such that the integer representations of all the valid outputs $P_1, P_2, \ldots, P_m$ taken mod q are all distinct, i.e, P_i mod $q = w_i$ and all w_i's are distinct. The w_i's can be represented by $O(\log m)$ bits. Thus, the representation of all the valid outputs can be hashed to $O(\log m)$ and hence $O(\log n)$ bits without any collisions.

To find q, queries are sent to an oracle B^c, where B is defined as:

$$\begin{aligned} B = \{< M, x, q >| & \text{ on input } x \text{, there are at least two accepting} \\ & \text{paths of M whose outputs are distinct but when taken mod } q, \\ & \text{have the same value } \}. \end{aligned}$$

B lies in NL. (The corresponding NL machine guesses two paths and computes the mod q of their outputs in parallel and verifies whether the mods are equal and the paths are accepting.)

Now, the machine does the following:

$w_{max} = 1$
for $i := 1$ to $q - 1$
  begin
    if $< M, x, i, w_{max} >\in C$
      $w_{max} = i$
  end
output w_{max}

where C is defined as:

$$C = \{< M, x, w_i, w_j >| \; w_i \text{ and } w_j \text{ represent valid outputs and the output represented by } w_i \text{ is greater than that by } w_j\}.$$

Here C lies in NL. The corresponding NL machine guesses two paths and verifies in parallel that the mods of their outputs are w_i and w_j and at the end verifies that both the paths are accepting and the output represented by w_i is greater than that by w_j.

Once the L-transducer gets w_{max}, it queries another oracle D to produce the final output. Here D is defined as follows:

$$D = \{< M, x, w_{max}, i, b >| \text{ on input } x, \text{ the i'th bit of the valid output of M with representation } w_{max} \text{ has value b } \}$$

It is not hard to see that D lies in NL. The L-transducer obtains every bit of output by making l(the number of bits in the output is exactly l) queries to oracle D.

∎

Theorem 3.3 $\#L_p \subseteq FL(NL)$.

Proof: The proof of this theorem can easily be inferred by slight modification of the proof of the earlier theorem 3.2. Let M be an NL machine with at most polynomial number of accepting paths. It can easily be seen that each accepting path of M can be represented in binary; the representations are bounded by a polynomial. By Lemma 3.1, we can find a prime p such that the integer representation of each of the path taken mod p gives a different residue. Once the prime number p is found, we can cycle through the residues, i.e, the numbers till p and check for each residue whether a corresponding accepting path exists or not; if it does, we should increment the counter by 1 (since the number of accepting paths is bounded by a polynomial, a counter can be maintained in logspace), else do nothing and move over to the next residue. The value of the counter at the end (when the cycle is complete) gives the value of the $\#L_p$ function for the machine M. ∎

Chapter 4

StrongFewL and UL

4.1 Relation between $StrongFewL$ and UL

The following lemma is proved in [RA97].

Lemma 4.1 *There is a logspace computable function f and a sequence of "advice strings" $\alpha(n) \mid n \in \mathbf{N}$ (where $\mid \alpha(n) \mid$ 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 a s-t path then there is some i such that G_i is a min-unique graph.*

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

- If G_i is not min-unique, M has a unique path that determines this fact and goes onto process G_{i+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.

This routine gives us the following lemma:

Lemma 4.2 *Given a sequence of graphs $\langle G_1, \ldots, G_k \rangle$, satisfying the properties of Lemma 4.4 , there exists a UL algorithm to determine s-t connectivity.*

The proof of Lemma 4.2 is given in the appendix.

Combining the construction of Lemma 4.1 and Lemma 4.2 we get the following theorem:

Theorem 4.3 $NL \subseteq UL/poly$

However, the need of "advice strings" in the Lemma 4.1 can be removed in the case of the directed, layered graphs having at most polynomial number of paths from s to any vertex in the graph. Such graphs represent the configuration graphs of $StrongFewL$ machines. We will call such graphs **strong few graphs** for convenience. Our next lemma precisely does this derandomization.

Lemma 4.4 *There is a logspace computable function f with the following properties:*

- *For any* **strong few graph** *G on n vertices, $f(G) = \langle G_1, \ldots, G_{p(n)} \rangle$.*
- *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 a s-t path then there is some i such that G_i is a min-unique graph.*

Proof: As the graph G is layered there can only be edges directed from vertices at layer i to the vertices at layer $i + 1$. We assume that the out degree of any vertex in the graph G is at most 2. Given the graph G our logspace computable function f returns a sequence of weighted graphs $\langle G_1, \ldots, G_{p(n)} \rangle$; the weights are assigned to the edges of graph G_q in the following way:

- The edge from a vertex v at layer i is assigned weight 0 if the out degree of the vertex v is 1.
- If the out degree of a vertex v at layer i is 2 then the left edge is assigned weight 0 and the right edge is assigned a weight of $2^i \bmod q$ (the distinction between left and right edges can be done by considering the ordering of the vertices).

It is obvious that graph G_q has an s-t path if and only if G has an s-t path (we are not adding or removing any edge). Combining the result of Lemma 3.1 and the fact that graph G has only polynomial number of paths from s to any other vertex, we can see that there exists a prime p such that the weight of any path from s to any other vertex of graph G_p is distinct. The value $p(n)$ can be chosen as in Lemma 3.1. Thus there exists a min-unique graph in the sequence output by the function f. ∎

Combining this Lemma 4.4 and the construction of Lemma 4.2 we get the following theorem:

Theorem 4.5 $StrongFewL \subseteq UL$

4.2 Relation between $ConstL$ and UL

Theorem 4.6 *If* $UL =$ *co-*UL *then* $ConstL \subseteq UL$.

Proof: First of all we show that there is a UL machine M which when given input $\langle G, c \rangle$, where G is a digraph with the assurance that it has at most c paths from s to t, it will accept if and only if G has exactly c paths from s to t. For this we consider the **critical subgraph G'** of G. This graph G' has either a constant number of paths or no path from s to t. So G' has constant number of nodes and hence can be represented using logspace. The UL machine M guesses this **critical subgraph** and verifies in logspace that it is the **critical subgraph** of G and that it has c number of paths from s to t. If M wrongly guesses the **critical subgraph** then M will obviously reject. It will also reject in the following two cases:

- M guesses the **critical subgraph** correctly but the number of paths from s to t is less than c.
- M guesses a subgraph of G' (in this case, the verification that it has c paths from s to t will fail).

Hence the machine M will accept unambiguously if and only if G has c paths from s to t.

Now if $UL = \text{co-}UL$ then there exist another UL machine M^c which will accept if and only if M rejects and vice versa.

Input (G)
for $c' := c$ to 1 **do** (* where c is a large constant *)
  begin
    guess if M accepts $\langle G, c' \rangle$
    if guess is *yes* **then**
      begin
        Run M on $\langle G, c' \rangle$
        if M accepts $\langle G, c' \rangle$ **then**
          Accept and halt
        else reject
      end
    else (* guess is *no* *)
      begin
        Run M^c on $\langle G, c' \rangle$
        if M^c accepts $\langle G, c' \rangle$ **then**
          continue forloop
        else reject
      end
  end
endfor
if $c' = 1$ **then** reject.

We run the above routine. It can be easily seen that the routine accepts unambiguously if and only if there exists a s-t path in G and rejects otherwise.

∎

Chapter 5

Conclusions

We have studied the counting and the optimization version of nondeterministic logspace machine and also the unambiguous version of the NL class. We derived important results for the NL machines with the restriction that the number of accepting paths of the machine is at most polynomial; the $OptL$ and the $\#L$ functions have been shown to be contained in the $FL(NL)$ class for this machine.

Allender and Reinhardt [RA97] have used randomization to show that the NL machine may be made unambiguous. We worked to derandomize the algorithm and have shown that the NL machine under certain restrictions, i.e, the $StrongFewL$ machine can be made unambiguous. Also under the assumption that $UL = co{-}UL$, it is shown that the NL machine with at most a constant number of accepting paths can also be made unambiguous. This gives rise to the natural question whether UL $= NL$.

5.1 Suggestion for future work

A lot of interesting open questions still remain to be answered and should be delved into in the future.

- It has been shown that $OptL$ class is contained in the $FL(NL)$ class for the NL machine having at most polynomial number of accepting paths. Alvarez and Jenner [AJ93] have shown $FL(NL)$ to be contained in $OptL$. Thus it is a natural question whether $OptL = FL(NL)$.

- It is a natural question is whether the randomized aspect of the construction given by Allender and Reinhardt [RA97] can be removed to obtain $UL = NL$. In fact there are other related questions like is $UL = FewL$ and $UL = co{-}UL$.

Appendix A

A.1 Inductive double counting method

This method is given by Reinhardt and Allender, [RA97]; it is given in this appendix for the sake of completeness.

Lemma A.1 *Given a sequence of graphs $\langle G_1, \ldots, G_k \rangle$, satisfying the properties of Lemma 4.4 , there exists a UL algorithm to determine s-t connectivity.*

Proof:

The UL machine M processes each graph G_i in sequence, uniquely determines whether G_i is min-unique with a s-t path or not; if G_i is not min-unique, it moves on to G_{i+1}, if G_i is a min-unique with a s-t path, it accepts it unambiguously while, if G_i has no s-t path, M has no accepting path.

The technique used here is known as the *double counting method* 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, these numbers are denoted by c_k and $\sum_k$ respectively. Let us use the notation $d(v)$ to denote the length of the shortest path in the graph G from the start vertex to v. (If no such path exists, then $d(v) = n+1$.)Thus using this notation, $\sum_k = \sum_{\{x|d(x)\leq k\}} d(x)$.
A useful observation is that *if the subgraph of G having a distance at most k from the start vertex is min-unique* (and if the correct values of c_k and $\sum_k$ are provided), then an unambiguous logspace machine can on input $(G, k, c_k, \sum_k, v)$ compute the

boolean predicate "$d(v) \leq k$". This is achieved by the routine shown below.

Input $(G, k, c_k, \sum_k, v)$
count := 0; *num* := 0; *path.to.v* := *false* ;
for each x $\in V$ **do**
  Guess nondeterministically if $d(x) \leq k$.
  if the guess is $d(x) \leq k$ then
    begin
    Guess a path of length $l \leq k$ 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* $= \sum_k$
  then return the Boolean value of *path.to.v*
  else halt and reject
end.procedure

To see that the routine is unambiguous, note the following :

- 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 then $\sum_k$, 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 $\sum_0 = 0$. A key part of the construction involves computing

c_k and $\sum_k$ from c_{k-1} and $\sum_{k-1}$, 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 min-unique if and only if there exist some v and x as above, as well as some other $x' \neq x$ such that $d(x') \leq k-1$ and there is an edge (x', v). The formal code is given below :

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

```
    c_k := c_{k-1}; Σ_k := Σ_{k-1};
for each  vertex v do
        if ¬(d(v) ≤ k − 1) then
            for each x such that (x, v) is an edge do
                if d(x) ≤ k − 1 then
                    begin
                    c_k := c_k + 1; Σ_k := Σ_k +k;
                    for x' ≠ x do
                        if (x, v) is an edge and d(x') ≤ k − 1
                            then  BAD.GRAPH := true
                    endfor
                    end
            endfor
endfor At this point, the values of c_k and Σ_k are correct.
```

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}; 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.

Input (G)
$BAD.GRAPH := false$; $c_0 := 1$; $\sum_0 := 0$; $k := 0$
repeat
    $k := k + 1$
    compute c_k and $\sum_k$ from $(c_{k-1}, \sum_{k-1})$
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 $d(t) \leq k$.

∎

Bibliography

[AJ93] Alvarez and Jenner, A very hard log-space counting class Theoretical Computer Science 107 (1993)3-30

[AJ95] Alvarez and Jenner, A Note on Logspace Optimization Computational Complexity 5 (1995), 155-166

[RA97] K. Reinhardt and E. Allender, Making Nondeterminism Unambiguous. In 38th IEEE Symposium on Foundations of Computer Science (FOCS), page 244-253, 1997.

[BJLR] G. Buntrock, B Jenner, K, J Lange, Rossmanith, Unambiguity and fewness for logarithmic space, vol 529 of Lecture Notes in Computer Science,168 - 179.

[Pap] Papadimitrou', Computational complexity.