Electronic Colloquium on Computational Complexity
ECCC Technical Reports Series 1994 Report Nr: TR94-003

Symmetric Logspace is Closed Under Complement

Noam Nisan[†] Amnon Ta-Shma[‡]

Received November 6 1994

Abstract. We present a Logspace, many-one reduction from the undirected st-connectivity problem to its complement. This shows that $SL = co - SL$.

[†] Institute of Computer Science, Hebrew University of Jerusalem. email: noam@cs.huji.ac.il. This work was supported by BSF grant 92-00043 and by a Wolfeson award administered by the Israeli Academy of Sciences. The work was revised while visiting BRICS, Basic Research in Computer Science, Centre of the Danish National Research Foundation.
[‡] Institute of Computer Science, Hebrew University of Jerusalem. email: am@cs.huji.ac.il. This work was supported by BSF grant 92-00043 and by a Wolfeson award administered by the Israeli Academy of Sciences. The work was revised while visiting BRICS, Basic Research in Computer Science, Centre of the Danish National Research Foundation.

1 Introduction

This paper deals with the complexity class symmetric Logspace, SL, defined by Lewis and Papadimitriou in [LP82]. This class can be defined in several equivalent ways:

1. Languages which can be recognised by symmetric nondeterministic Turing Machines that run within logarithmic space. See [LP82].
2. Languages that can be accepted by a uniform family of polynomial size contact schemes (also sometimes called switching networks.) See [Raz91].
3. Languages which can be reduced in Logspace via a many-one reduction to $USTCON$, the undirected st-connectivity problem.

A major reason for the interest in this class is that it captures the complexity of $USTCON$. The input to $USTCON$ is an undirected graph G and two vertices in it s, t, and the input should be accepted if s and t are connected via a path in G. The similar problem, $STCON$, where the graph G is allowed to be directed is complete for NL, non-deterministic Logspace. Several combinatorial problems are known to be in SL or $co-SL$, e.g. 2-colourability is complete in $co-SL$ [Rei82].

The following facts are known regarding SL relative to other complexity classes in "the vicinity":

$$L \subseteq SL \subseteq RL \subseteq NL.$$

Here, L is the class deterministic Logspace and RL is the class of problems that can be accepted with one-sided error by a randomized Logspace machine running in polynomial time. The containment $SL \subseteq RL$ is the only non-trivial one in the line above and follows directly from the randomized Logspace algorithm for $USTCON$ of [AKL$^+$79]. It is also known that $SL \subseteq SC$ [Nis92], $SL \subseteq \bigoplus L$ [KW93] and $SL \subseteq DSPACE(\log^{1.5} n)$ [NSW92].

After the surprising proofs that NL is closed under complement were found [Imm88, Sze88], Borodin et al [BCD$^+$89] asked whether the same is true for SL. They could prove only the weaker statement, namely that $SL \subseteq co-RL$, and left "$SL = co-SL$?" as an open problem. In this paper we solve the problem in the affirmative by exhibiting a Logspace, many-one reduction from $USTCON$ to its complement. Quite surprisingly the proof of our theorem does not use inductive counting, as do the proofs of $NL = co-NL$, and is in fact even simpler than them, however it uses the [AKS83] sorting networks.

Theorem 1 *$SL = co-SL$.*

It should be noted that the monotone analogues (see [GS91]) of SL and $co-SL$ are known to be different [KW88].

As a direct corollary of our theorem, we get that $L^{SL} = SL^{SL} = SL$ where L^{SL} is the class of languages accepted by *Logspace* oracle Turing machines with oracle from SL, and SL^{SL} is defined similarly, being careful with the way we allow queries (see [RST84]).

Corollary 1.1 $L^{SL} = SL^{SL} = SL$

In particular this shows that both "symmetric Logspace hierarchies", (the one defined by alternation in [Rei82], and the one defined by oracle queries in [BALPS94]) collapse to SL.

2 Proof of Theorem

2.1 Overview of proof.

We design a many-one reduction from $co-USTCON$ to $USTCON$. We start by developing, in subsection2.2, simple tools for combining reductions. In particular these tools will allow us to use the AKS sorting networks in order to "count". At this point, the main ingredient of the reduction will be the calculation of the number of the connected components of a graph. An upper bound to this number is easily obtained using tarnsitive closure, while the main idea of the proof is to obtain a lower bound by computing a spanning forest of the graph, which is done in subsection 2.3. In subsection 2.4 everything is put together.

2.2 Projections to $USTCON$.

In this paper we will use only the simplest kind of reductions, i.e. *LogSpace* uniform projection reductions [SV85]. Moreover, we will be interested only in reductions to $USTCON$. In this subsection we define this kind of reduction and we show some of its basic properties.

NOTATION 2.1 *Given* $f : \{0,1\}^* \mapsto \{0,1\}^*$ *denote by* $f_n : \{0,1\}^n \mapsto \{0,1\}^*$ *the restriction of* f *to inputs of length* n. *Denote by* $f_{n,k}$ *the* k*'th bit function of* f_n, *i.e. if* $f_n : \{0,1\}^n \mapsto \{0,1\}^{k(n)}$ *then* $f_n = (f_{n,1}, \ldots, f_{n,k(n)})$.

NOTATION 2.2 *We represent an* n*–node undirected graph* G *using* $\binom{n}{2}$ *variables* $\vec{x} = \{x_{i,j}\}_{1\leq i<j\leq n}$ *s.t.* $x_{i,j}$ *is* 1 *iff* $(i,j) \in E(G)$. *If* $f(\vec{x})$ *operates on graphs , we will write* $f(G)$ *meaning that the input to* f *is a binary vector of length* $\binom{n}{2}$ *representing* G.

We say that $f : \{0,1\}^* \mapsto \{0,1\}^*$ reduces to $USTCON(m)$ if we can (uniformly and in *LogSpace*) label the edges of a graph of size m with $\{0,1,x_i,\neg x_i\}_{1\leq i\leq n}$, s.t. $f_{n,k}(\bar{x}) = 1 \iff$ there is a path from 1 to m in the corresponding graph. Formally,

DEFINITION 2.1 *We say that* $f : \{0,1\}^* \mapsto \{0,1\}^*$ *reduces to* $USTCON(m)$ *,* $m = m(n)$, *if there is a uniform family of* $Space(log(n))$ *functions* $\{\sigma_{n,k}\}$ *s.t. for all* n *and* k*:*

- $\sigma_{n,k}$ *is a projection, i.e.:* $\sigma_{n,k}$ *is a mapping from* $\{i,j\}_{1\leq i<j\leq m}$ *to* $\{0,1,x_i,\neg x_i\}_{1\leq i\leq n}$
- *Given* $\vec{x}$ *define* $G_{\vec{x},k}$ *to be the graph* $G_{\vec{x},k} = (\{1,\ldots,m\},E)$ *where* $E = \{(i,j) \mid \sigma_{n,k}(i,j) = 1$ *or* $\sigma_{n,k}(i,j) = x_i$ *and* $x_i = 1$ *or* $\sigma_{n,k}(i,j) = \neg x_i$ *and* $x_i = 0\}$.
- $f_{n,k}(\vec{x}) = 1 \iff$ *there is a path from* 1 *to* m *in* $G_{\vec{x},k}$.

If σ *is restricted to the set* $\{0,1,x_i\}_{1\leq i\leq n}$ *we say that* f monotonically *reduces to* $USTCON(m)$.

Lemma 2.1 *If f has uniform monotone formulae of size $s(n)$ then f is monotonically reducible to $USTCON(O(s(n)))$.*

Proof: Given a formula ϕ recursively build (G, s, t) as follows:

- If $\phi = x_i$ then build a graph with two vertices s and t, and one edge between them labelled with x_i.
- If $\phi = \phi_1 \wedge \phi_2$, and (G_i, s_i, t_i) the graphs for ϕ_i, $i = 1, 2$, then identify s_2 with t_1 and define $s = s_1, t = t_2$.
- If $\phi = \phi_1 \vee \phi_2$, and (G_i, s_i, t_i) the graphs for ϕ_i, $i = 1, 2$, then identify s_1 with t_1 and s_2 with t_2 and define $s = s_1 = t_1$ and $t = s_2 = t_2$.

□

Using the AKS sorting networks [AKS83], which belong to NC^1 , we get:

Corollary 2.2 *$Sort : \{0,1\}^* \mapsto \{0,1\}^*$ (which given a binary vector sorts it) is monotonically reducible to $USTCON(poly)$.*

Lemma 2.3 *If f monotonically reduces to $USTCON(m_1)$ and g reduces to $USTCON(m_2)$ then $f \circ g$ reduces to $USTCON(m_1^2 \cdot m_2)$, where $\circ$ is the standard function composition operator.*

Proof: f monotonically reduces to a graph with m_1 vertices, where each edge is labelled with one of $\{0, 1, x_i\}$. In the composition function $f \circ g$ each x_i is replaced by $x_i = g_i(\vec{y})$ which can be reduced to a connectivity problem of size m_2. Replace each edge labelled x_i with its corresponding connectivity problem. There can be m_1^2 edges, each replaced by a graph with m_2 vertices, hence the new graph has $m_1^2 \cdot m_2$ vertices. □

2.3 Finding a spanning forest.

In this section we show how to build a spanning forest using $USTCON$. This basic idea was already noticed by Reif and independently by Cook [Rei82].

Given a graph G index the edges from 1 to m. We can view the indices as weights to the edges, and as no two edges have the same weight, we know that there is a unique minimal spanning forest F. In our case, where the edges are indexed, this minimal forest is the lexicographically first spanning forest.

It is well known that the greedy algorithm finds a minimal spanning forest. Let us recall how the greedy algorithm works in our case. The algorithm builds a spanning forest F which is at the beginning empty $F = V$. Then the algorithm checks the edges one by one according to their order, for each edge e if e does not close a cycle in F then e is added to the forest, i.e. $F = F \cup \{e\}$.

At first glance the algorithm looks sequential, however, claim 2.3 shows that the greedy algorithm is actually highly parallel. Moreover, all we need to check that an edge does not participate in the forest, is one st connectivity problem over a simple to get graph.

Definition 2.2 *For an undirected graph G denote by $LFF(G)$ the lexicographically first spanning forest of G. Let*

$SF(G) \mapsto \{0,1\}^{\binom{n}{2}}$ *be:*

$$SF_{i,j}(G) = \begin{cases} 0 & (i,j) \in LFF(G) \\ 1 & \textit{otherwise} \end{cases}$$

Lemma 2.4 *SF reduces to $USTCON(poly)$*

Proof: Let F be the lexicographically first spanning forest of G. For $e \in E$ define G_e to be the subgraph of G containing only the edges $\{e' \in E \mid index(e') < index(e)\}$.

Claim: $e = (i,j) \in F \iff e \in E \wedge i$ is not connected to j in G_e.

Proof: Let $e = (i,j) \in E$. Denote by F_e the forest which the greedy algorithm built at the time it was checking e. So $e \in F \iff e$ does not close a cycle in F_e.

($\Longrightarrow$) $e \in F$ and therefore e does not close a cycle in F_e, but then e does not close a cycle in the transitive closure of F_e, and in particular e does not close a cycle in G_e.

($\Longleftarrow$) e does not close a cycle in G_e therefore e does not close a cycle in F_e and $e \in F$. □

Therefore $SF_{i,j}(G) = \neg x_{i,j} \vee i$ is connected to j in $G_{(i,j)}$.

Since $\neg x_{i,j}$ can be viewed as the connectivity problem over the graph with two vertices and one edge labelled $\neg x_{i,j}$ it follows from lemmas 2.1, 2.3 that SF reduces to $USTCON$. Notice, however, that the reduction is not monotone.

□

2.4 Putting it together.

First, we want to build a function that takes one representative from each connected component. We define $LI_i(G)$ to be 0 iff the vertex i has the largest index in its connected component.

Definition 2.3 $LI(G) \mapsto \{0,1\}^n$

$$LI_i(G) = \begin{cases} 0 & \textit{i has the largest index in its connected component} \\ 1 & \textit{otherwise} \end{cases}$$

Lemma 2.5 *LI reduces to $USTCON(poly)$*

Proof:

$LI_i(G) = \bigvee_{j=i+1}^{n} \quad (i$ is connected to j in $G)$.

So LI is a simple monotone formula over connectivity problems, and by lemmas 2.1, 2.3 LI reduces to $USTCON$. This is, actually, a monotone reduction.

$\square$

Using the spanning forest and the LI function we can exactly compute the number of connected components of G, i.e.: given G we can compute a function NCC_i which is 1 iff there are exactly i connected components in G.

Definition 2.4 $NCC(G) \mapsto \{0,1\}^n$

$$NCC_i(G) = \begin{cases} 1 & \textit{there are exactly } i \textit{ connected components in } G \\ 0 & \textit{otherwise} \end{cases}$$

Lemma 2.6 *NCC reduces to $USTCON(poly)$*

Proof:

Let F be a spanning forest of G. It is easy to see that if G has k connected components then $|F| = n - k$.

Define:

$$f(G) = Sort \circ LI(G)$$
$$g(G) = Sort \circ SF(G).$$

Then:

$$\begin{array}{lll} f_i(G) = 1 & \Longrightarrow k < i & \\ g_i(G) = 1 & \Longrightarrow n - k < i & \Longrightarrow k > n - i. \end{array}$$

and thus: $NCC_i(G) = f_{i+1}(G) \wedge g_{n-i+1}(G)$

Therefore applying lemmas 2.1, 2.2, 2.3, 2.4, 2.5 proves the lemma.

$\square$

Finally we can reduce the non-connectivity problem to the connectivity problem, thus proving that $SL = co - SL$.

Lemma 2.7 *$\overline{USTCON}$ reduces to $USTCON(poly)$*

Proof:

Given (G, s, t) define G^+ to be the graph $G \cup \{(s,t)\}$.

Denote by $\#CC(H)$ the number of connected components in the undirected graph H.

$$\begin{array}{ll} s \text{ is not connected to } t \text{ in } G & \Longleftrightarrow \\ \# \, CC(G^+) = \# \, CC(G) - 1 & \Longleftrightarrow \\ \bigvee_{i=2,\ldots,n} \; NCC_i(G) \wedge NCC_{i-1}(G^+). & \end{array}$$

Therefore applying lemmas 2.1, 2.3, 2.6 proves the lemma. $\square$

3 Extensions

Denote by L^{SL} the class of languages accepted by *Logspace* oracle Turing machines with oracle from SL. An oracle Turing machine has a work tape and a write-only query tape (with unlimited length) which is initialised after every query. We get:

Corollary 3.1 $L^{SL} = SL$.

Proof:

Let $Lang$ be a language in L^{SL} solved by an oracle Turing machine M running in L^{SL}, and fix an input $\vec{x}$ to M.

Look at the configuration graph of M. In this graph we have query vertices with outgoing edges labelled "connected" and "not connected". We would like to replace the edges labelled "connected" with their corresponding connectivity problems, and the edges labelled "not connected" with the connectivity problems obtained using our theorem that $SL = co-SL$.

However, there is a technical problem here, as the queries are determined by the edges and not by the query vertices. We can fix this difficulty by splitting each query vertex to its "yes" and "no" answers, and splitting each edge entering a query vertex to "connected" and "not connected" edges. Now we can easily replace each edge with a connectivity problem, obtaining an undirected graph which is *st* connected iff $\vec{x} \in Lang$, and therefore $Lang \in SL$.

$\square$

As can easily be seen the above argument applies to any undirected graph with $USTCON$ query vertices, thus, if we carefully define SL^{SL} (see [RST84]) we get that:

Corollary 3.2 $SL^{SL} = SL$.

4 Acknowledgements

We would like to thank Amos Beimel, Allan Borodin, Assaf Schuster, Robert Szelepcsényi, and Avi Wigderson for helpful discussions.

References

[AKL+79] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovasz, and C. Rackoff. Random walks, universal sequences and the complexity of maze problems. In *Proceedings of the 20th Annual IEEE Symposium on the Foundations of Computer Science*, 1979.

[AKS83] M. Ajtai, J. Komlos, and E. Szemeredi. An $O(n \log n)$ sorting network. In *Proc. 15th ACM Symposium on Theory of Computing (STOC)*, pages 1–9, 1983.

[BALPS94] Y. Ben-Asher, K.-J. Lange, D. Peleg, and A. Schuster. The complexity of reconfiguring network models. Manuscript, 1994.

[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.

[GS91] Grigni and Sipser. Monotone separation of logspace from nc^1. In *Annual Conference on Structure in Complexity Theory*, 1991.

[Imm88] Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17, 1988.

[KW88] M. Karchmer and A. Wigderson. Monotone circuits for connectivity require super-logarithmic depth. In *Proc. 20th ACM Symposium on Theory of Computing (STOC)*, pages 539–550, 1988.

[KW93] Karchmer and Wigderson. On span programs. In *Annual Conference on Structure in Complexity Theory*, 1993.

[LP82] Lewis and Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19, 1982.

[Nis92] N. Nisan. RL $\subseteq$ SC. In *Proc. 24th ACM Symposium on Theory of Computing (STOC)*, pages 619–623, 1992.

[NSW92] N. Nisan, E. Szemeredi, and A. Wigderson. Undirected connectivity in $O(log^{1.5}n)$ space. In *Proc. 33th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 24–29, 1992.

[Raz91] A. Razborov. Lower bounds for deterministic and nondeterministic branching programs. In *Proceedings of the 8th FCT, Lecture Notes in Computer Science,* 529, pages 47–60, New York/Berlin, 1991. Springer-Verlag.

[Rei82] J. H. Reif. Symmetric complementation. In *Proc. 14th ACM Symposium on Theory of Computing (STOC)*, pages 201–214, 1982.

[RST84] Ruzzo, Simon, and Tompa. Space-bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28:216–230, April 1984.

[SV85] Skyum and Valiant. A complexity theory based on boolean algebra. *Journal of the ACM*, 1985.

[Sze88] Szelepcsenyi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26, 1988.