Universality for Nondeterministic Logspace[*]

Vinay Chaudhary, Anand Kumar Sinha, and Somenath Biswas
Department of Computer Science and Engineering
IIT Kanpur[†]

September 2004

1 Introduction

The notion of universality was introduced [AB92] in 1992 in the context of **NP**. The major motivation of the work was to capture in a theory the essential sameness of various NP-completeness proofs. The approach was to find out precisely what is preserved by natural reductions. When we wish to show that a language A polynomial time, many-one reduces to an NP-complete language B vis a reduction f, the reduction f needs only to preserve membership, viz., $x \in A$ iff $f(x) \in B$. However, natural reductions can be seen to preserve far more than membership. The notion of universality provided precisely what remains preserved: it is the set of witnesses or solutions each witnessing $x \in A$ (in the context of an NP relation R_A defining the language A) that remains preserved, in the sense that from the set of witnesses witnessing $f(x) \in B$ (in the context of an NP relation R_B defining B) the set of witnesses witnessing $x \in A$ can be extracted in a feasible manner. Finally, [AB92] also gave a structural characterization of universal NP relations.[1]

Further to [AB92], Portier extended universality for NP-completeness over certain algebraic structures [P98]. In 1994, Buhrman et. al. [BKT94] made use of NP universality to provide sufficient conditions for NP optimization problems that admit efficient approximation algorithm. Chakraborty and Kumar [CK03] extended the notion of universality for completeness in #**P**in 2003. Fournier and Malod [FM03] subsequently provided a cleaner treatment of #P universality. In the current paper, we consider NL universality.[2]

[*]Work reported here has been partially supported by IFCPAR Project 2602 – 1

[†]The first two authors are currently at the Computer Science Department, University of Wisconsin, Madison.

[1]As we shall see later that universality is defined as an attribute of a relation that defines a language, rather than as an attribute of the language *per se*. If the relation is universal then the corresponding language is guranteed to be complete.

[2]The work reported here is a revised version of a part of the BTech project work of Chaudhary and Sinha

NL is the class of languages which are accepted by nondeterministic Turing machines using logarithmic workspace. It is well known that this class can be alternatively defined also as: (1) the class of languages accepted by nondeterministic counter machines having a constant number of counters where each counter can count only upto the length of the input string, or as (2) the class of languages accepted by nondeterministic machines with a fixed number of read-only, two-way input heads which cannot move beyond the input, these machines have no write capability, and hence no workspace.

Languages accepted (and functions computed) with logarithmic space usage constraint are of interest for various reasons. First, we know that

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$$

and that at least one of the four inclusions above must be strict, but we do not know which one(s). We would like to know, therefore, how **L** relates to **NL**, and as well as how these relate to **P**. Next, logspace restricted computations, even deterministic ones, are fairly powerful: it is known that a fair amount of arithmetic can be done in deterministic logspace,[3] and all known natural reductions to NP-complete sets can be carried out within this resource bound. The emerging importance streaming algorithms has also made the question of what is computable in logspace topical.

Two reasons motivated us to investigate universality for NL. First, NP universality is obtained in terms of NP relations that define NP languages. NL languages too can be defined in terms of relations, and therefore, work similar to NP can indeed be done for NL. The other reason is: we know from the Immerman-Szelepcsényi result [I88] that NL is closed under complementation, and therefore, the complement of every NL-complete language is also complete for NL. Thus, it will be interesting to see how the notion of universality that we develop for NL will work equally well, for example, for directed graph s-t connectivity[4] *as well as* for directed graph s-t *un*connevtivity.

We cite [JLL76] for examples of NL-complete languages from various domains, the complements of all these languages are, of course, also NL-complete.

Section 2 gives the definitions that we use. Next, in Section 3 we provide a structural characterization of NL-universality, and in Section 4 we show how directed graph s-t unconnectivity also satisfies this characterization.

[CS04].

[3] An interesting recent discovery is that division can be carried out within this resource.

[4] which is the canonical complete problem for NL

2 Basic Concepts

2.1 Relations, their associated languages, and solutions

Let $R \subseteq \Sigma^* \times \Sigma^*$ be a relation. The language $L_R \subseteq \Sigma^*$, *associated* with the relation R is defined as

$$L_R \stackrel{\text{def}}{=} \{x | (\exists y)(x, y) \in R\}$$

When $x \in L_R$, and $(x, y) \in R$, we call y to be a *solution* that witnesses x to be in L_R.

The set of all solutions of x (with respect to the relation R), denoted as $sol_R(x)$, we have, therefore, $sol_R(x) = \{y | (x, y) \in R\}$.

Throught we make the following assumption:
Every element of $\text{sol}_R(x)$ *has the same length, denoted as* $\text{sol-len}_R(x)$ *and if* $|x_1| = |x_2|$ *then* $\text{sol-len}_R(x_1) = \text{sol-len}_R(x_2)$.

By restricting ralation R in various ways, we get the associated language L_R to belong to different complexity classes. For example, when the graph of R is a language in **P** and there is a fixed polynomial $p(\cdot)$ such that if $x \in L_R$ then $\textit{sol-len}_R(x) = p(|x|)$, then $L_R \in \mathbf{NP}$. Such a relation R is called an NP relation.

2.2 NL relations

Definition 1 *A relation R is an NL relation if it satisfies the following conditions:*

1. *There is a fixed polynomial $p(\cdot)$ such that whenever x is in L_R,* $\text{sol-len}_R(x) = p(|x|)$, *and*

2. *There is a deterministic logspace machine M_R with two read only input tapes, one of which is two-way, the other being one-way such that , for all (x, y), with x on the two-way input tape, and y on the one-way tape, M_R accepts iff $(x, y) \in R$.*

Proposition 2 *A language L is in NL iff there is an NL relation R such that $L = L_R$.*

The restriction that a solution y be placed in a one-way input tape of a logspace machine is necessary to prove the only if part of the above proposition.

2.3 Blocks, block-masks, and projections through masking

Although, we can think of solutions as binary strings without loss of generality, often we will require to view solutions in the following, more structured, fashion. We will assume that if R is an NL relation, then every solution of length n is the concatenation of a number of equal length *blocks*. Length of a block in a solution of x is $O(\log x)$. If $x \in L_R$, then *block-len*(x) specifies the length of each block in solutions of x, and further, *block-len*$(\cdot)$ is logspace computable.(Clearly, *sol-len*(x) is an integral multiple of *block-len*(x)).

Typically, a block will be the code of an element of some set, or the label of a vertex in a graph, etc.

We now explain what a *block-mask* is, the purpose of a block-mask is to filter out certain bits from a block. Let l be the length of each block for some x. A block-mask then is also a string of length l over $\{r, d, m_1, m_0\}$. The ith symbol of the block-mask specifies what is to be done with the ith bit of a block: r is for retaining that bit, d is for dropping that bit, m_1 for matching the bit with 1 and m_0 is for matching with 0. When a block-mask is applied on a block, the bits of the block are retained or dropped as specified in the mask *provided* bits those are to be matched as specified by m_0 or m_1 in the mask are indeed matched in the block. If this matching is not successful, then the result of applying the mask on the block is an empty sequence. Matched bits, if any, are not retained on masking.

We will refer to symbols d and r in a mask as selection symbols, and m_1 and m_0 as match symbols.

We clarify the idea of masking with an example. Suppose that the block length is seven, and let a block-mask α be$rdrm_1dm_0r$. Consider a block 0010110. When α is applied on this block, the result is the null sequence, as the fourth and the sixth bits of the block are not 1 and 0 respectively. On the other hand, for the block 0011101, the result of applying α is 011.

Definition 3 (Projection through masking) *Let S be a set of solutions where the block length is l, and each solution be of m blocks. Let α be a block-mask. Then the projection of S through α, denoted by* $\mathrm{proj}_\alpha(S)$ *is defined as the set*
$\{\alpha(b_1)\alpha(b_2)\cdots\alpha(b_m)|$ *Each b_i is a block, and $b_1b_2\cdots b_m \in S$*$\}$

Definition 4 (Admissible block-masks) *A block-mask α is said to be admissible for a set S of solutions if*

1. *The length of α is same as that of a block in the solutions,*
2. *α has at least one r symbol, and*
3. *If match symbols m_0 or m_1 occur in α, then in every solution in S there will exist at least one block where matching will occur on every match symbol in α.*

The proposition below readily follows from the relevant definitions. We will see subsequently that this is a key fact used in many of our results.

Proposition 5 *For any relation R, a string x, and a block-mask α, such that α is admissible for* $\mathrm{sol}_R(x)$, $\mathrm{proj}_\alpha(sol_R(x))$ *is non-empty iff $x \in L_R$.*

2.4 Solution preserving reductions, NL universility

Now we define the notion of *solution preserving reductions* which is used to define universality.

Definition 6 *$f : \Sigma^* \to \Sigma^*$ is said to be a solution preserving reduction of an NL relation Q to an NL relation R if*

1. *f is computable in logspace, and*
2. *for all $x \in \Sigma^*$ there exist some string z and a block-mask α admissible for* $\mathrm{sol}_R(z)$ *such that $f(x) = \langle z, \alpha \rangle$ satisfying* $\mathrm{proj}_\alpha(\mathrm{sol}_R(z)) = \mathrm{sol}_Q(x)$.

(We make the assumption that the pairing function used in the definition above is such that its inverse is computable in logspace).

Using Proposition 5, we get

Proposition 7 *If f is a solution preserving reduction of NL relation Q to NL relation R, then $L_Q \leq_m^{\log} L_R$ via an f' readily obtained from f.*

Definition 8 (NL-universality) *An NL relation R is NL universal if for every NL relation Q there is a solution preserving reduction of Q to R.*

We have, therefore,

Proposition 9 *If R is NL universal then L_R is NL-complete.*

Although it appears, as we shall see later, that for all known NL-complete languages, natural NL relations that define these languages are NL universal, the converse of the above Proposition will be a major breakthrough. Because, it will immediately imply that $\mathbf{L} \neq \mathbf{NL}$. Because, if **NL** collapses to **L** then the trivial language $\{1\}$ will be NL-complete, but no NL relation defining this trivial language can have a solution preserving reduction from an NL-relation Q with L_Q the set of solution set of which is infinite.

The immediate question that arises is if there are NL-universal relationis. Indeed, the natural NL relation defining the canonical NL-complete problem, viz., directed graph connectivity, is NL-universal. Next definition recalls this language, usually named as STCONN.

Definition 10 *STCONN* $= \{\langle G, s, t\rangle | G$ *is a directed graph, s and t are two distinguished vertices, and there is a path from s to t in G*$\}$.

The NL relation we use for STCONN is R_{STCONN}, and it is defined as:
$(\langle G, s, t\rangle, y) \in R_{\text{STCONN}}$ iff $y = v_{i_1}v_{i_2}\dots v_{i_n}$ where G has n vertices, v_{i_j}'s are vertex labels of G, v_{i_1} is s, and there is some k such that for all $j, 1 \le j < k$, $(v_{i_j}, v_{i_{j+1}})$ is an edge of G, and $v_{i_k} = v_{i_{k+1}} = \dots = v_{i_n} = t$. In other words, y gives the label of some $s-t$ path in G with the end padded with repetitions of the vertex label t so that y has exactly n vertex labels.
(We need this padding to make every solution of $\langle G, s, t\rangle$ to have the same length).

Theorem 11 *R_{STCONN} is NL universal.*

Proof: We prove this by showing that for every NL relation Q, there is a solution preserving reduction of Q to R_{STCONN}.

Let M be an NL machine that, given $x \in L_Q$ as input, guesses bit-by bit a solution y of x (w.r.t. the NL relation Q)[5] and verifies that $(x, y) \in Q$. We assume M to be such that whatever guess bit is chosen, it immediately appears as the last bit in the workspace of M.

For any input x, let l_x be the length of every configuration of M on x, let us assume that the workspace contents come at the end of the configuration, therefore, the guess bit chosen at a configuration appears as the last bit of the immediately next configuration. We note that given x, l_x is logspace computable. Let START$_x$ be the start configuration of M on x, (from which there are two possible transitions, one to one to a state with guess bit 1, the other with guess bit 0). Let M be such that it goes to a unique accepting configuration, ACCEPT, if it does accept x. Let us assume that the state label appears in the beginning of every configuration, and this labelling ensures that states of START$_x$, ACCEPT, as well as all states that are required to achieve the uniqueness of ACCEPT, (e.g., when M is blanking out the workspace, moving the heads to the leftmost positions) all start with 0, and rest of the states start with 1.

Consider the following logspace transducer T. On input x, first it cycles through all possible configurations of M on x, and outputs as the labels of vertices of the graph of configurations G_x. Next, T cycles through all pairs of these configurations, and outputs pairs (A, B) if M can go in a single step from configuration A to B. These are the edges of G_x. Then, T outputs START$_x$ and ACCEPT.

Next, T outputs the block-mask α. The length of α is l_x, this block-mask is such that it retains the last bit of a block (which is the guess bit used to arrive at the configuration that corresponds to this block from the previous configuration), and selects a block for projecting only if its first bit is 1.

[5]Without loss of generality we are assuming here that solutions of strings in L_Q are binary strings.

As M accepts x using a solution y as guess iff there is a path from START_x to ACCEPT in G_x, and y is the concatenation of the bits chosen by α. Hence, T computes a solution preserving reduction from Q to R_{STCONN}.

■

3 Characterization of NL universality

3.1 *build* and *prune* operations

We show in this section that an NL relation is universal iff it admits two operations, called here *build*, and *prune*. The operation *build* computes strings with certain specific solution spaces, and *prune* can be used to appropriately prune these solution spaces. We will prove that if these two operations exist for an NL relation R, then there will exist a solution preserving reduction from R_{STCONN} to R, and therefore, R is universal. For the other direction, first we will see that for R_{STCONN}, both *build* and *prune* exist. Next, we show that if a solution preserving reduction exists from R_{STCONN} to an NL relation R, then it is possible to define *build* and *prune* of R from the corresponding operations of R_{STCONN}.

Definition 12 (build operation) *An NL relation R is said to have the* build *operation if there is a logspace computable function* $\mathrm{build}_R : 1^* \to \Sigma^*$ *such that for all positive integers n, there will exist a string x over the alphabet of Σ, and a block-mask α, this block-mask is admissible for* $\mathrm{sol}_r(x)$, *such that* $\mathrm{block}_R(1^n) = \langle x, \alpha \rangle$ *satisfying the following:*

$$\mathrm{proj}_\alpha(\mathrm{sol}_R(x)) = \bigcup_{k=1}^{n-2} \bigcup_{P,|P|=k,P\subseteq\{2,\ldots,n-1\}} \bigcup_{\sigma\in S_k} c_1 \cdot c_{\sigma(p_1)} \cdot \ldots \cdot c_{\sigma(p_k)} \cdot c_n \cdot \ldots \cdot c_n$$

In the above, S_k denotes all the permutations of k elements, p_i denotes the ith element of P (in canonical ordering), (P being a set of k integers), c_i denotes the binary encoding of the positive integer i, and finally, in the argument string, at the end there will be exactly as many c_n's so that the total number of c_i's in the argument becomes exactly n. (We have used $\cdot$ to denote the string concatenation operation).

The definition captures (rather in a formidable manner!) the following property of the NL relation R. For every n, we can in logspace find x and α such that $proj_\alpha(sol_R(x))$ is precisely the set of the solutions with respect to R_{STCONN} of the complete directed graph of n vertices, where the vertices of the graph are labelled with the binary encodings of numbers 1 to n, 1 being the distiguished s vertex and n being the distinguished t vertex of STCONN.

Our next definition defines the *prune* operation.[6]

Definition 13 *We say that an NL relation R has a* prune *operation if there exists a logspace computable function* $\text{prune}_R : \Sigma^* \to \Sigma^*$ *such that for all x, and for all admissible block-masks α, and for every set of pairs*
$\{(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)\}$*, all u_i, v_j's being equal length strings, there exist z and an admissible block-mask β such that*

$$\text{prune}_R(\langle x, \alpha, \{(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)\}\rangle) = \langle z, \beta\rangle$$

with z and β satisfying

$\text{proj}_\beta(\text{sol}_R(z)) = \{w | w \in \text{proj}_\alpha(\text{sol}_R(x))$ *with the property that if w is the blocked projection (with α) of the solution y of x, where $y = b_1 b_2 \cdots b_r$, each b_i being a a block of y, then there exists no $j, 1 \le j < r$ such that for some i, $1 \le i \le k$, the string $u_j v_j$ is equal to* $\alpha(b_j)\alpha(b_{j+1}\}$

Basically, *prune* is a way of pruning solution space in the following sense. Consider the set S' of solutions of x which are all the solutions *except* those where the projections (with α) of two successive blocks results as the string $u_i v_i$, for some i. Now, z is a string whose solution set, when projected with β, is precisely the pruned solution set S' of x.

It is obvious that R_{STCONN} has *build*. We argue that it has *prune* too.

Proposition 14 *R_{STCONN} has* prune *operation.*

Proof: Suppose, we need to compute $\textit{prune}_{R_{\text{STCONN}}}$ on the argument
$(\langle G, s, t\rangle, \alpha, \{(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)\}\rangle)$. Recall the standard input format for $\langle G, s, t\rangle$: first the vertices of G are listed, then the edges in the form of pairs of vertices, and finally, s and t appear on the input.

Consider a logspace transducer T which first copies the set of vertices of G onto the output tape. Then it examines each edge (u, v) one after another. T will copy this edge onto the output tape iff $\alpha(u)\alpha(v) \ne u_i v_i$ for any i. After T has considered all the edges of G, it outputs s, and t, and finally, copies α onto the output tape. This concludes description of T which computes $\textit{prune}_{R_{\text{STCONN}}}$.

Logspace complexity of T is obvious. A witness for $\langle G, s, t\rangle$ is a sequence of vertices of G which traces out an $s - t$ path. Deleting all edges (u, v) from G such that $\alpha(u)\alpha(v) = u_i v_i$

[6]There is a problem in the definition as given here: the two strings u_i and v_i of a pair can come together not from successive blocks of a solution, but from distant blocks becuse the intermediate blocks are masked out with α. This can be fixed by insisting in the definition that *prune* needs to remove only those solutions where u_i and v_i are from successive blocks. However, there may be a less restrictive condition which will suffice.

for any i is necessary and sufficient to ensure that the resultant graph solution set will satisfy the *prune* requirements.

$\blacksquare$

3.2 Characterization theorem for NL universality

The Theorem below structurally characterizes universal NL relations.

Theorem 15 *An NL relation R is NL universal iff R has the two operations,* build *and* prune*.*

Proof: **Proof of $\Leftarrow$ direction:**

We prove the only if part by showing that if an NL relation has *build* and *prune* then we shall be able to construct a solution preserving reduction of R_{STCONN} to R.

First, notice that in logspace we can transform an arbitrary instance $\langle G, s, t\rangle$ of STCONN, G having n vertices, to $\langle G', 1, n\rangle$ where G' is an isomorphic copy of G with the vertices of G' being labelled with binary encodings of 1 to n, and with s mapped to 1 and t mapped to n. [7] We assume then, w.l.o.g., that we have an instance $\langle G, 1, n\rangle$ where vertices of G are 1 through n. Consider the following logspace transducer T_1. Given $\langle G, 1, n\rangle$, T_1 computes $build_R(1^n)$ which is, say, $\langle x, \alpha\rangle$. We have that $proj_\alpha(sol_R(x)$ is the set of all $1-n$ paths of a complete graph with n vertices, labelled with 1 to n. First part of T_1's output is x and the second part is α. T_1 also computes the set S which is $E_1 - E_2$, E_1 denoting the set of edges in the complete graph with n vertices labelled as mentioned, and E_2 denoting the edges in the input graph G to T. This S is the third part of T_1's output. Now, let T_2 be a logspace transducer that applies $prune_R$ to the output of T_1. The composition of T_1 followed by T_2 provides[8] a solution preserving reduction of R_{STCONN} to R which directly follows from the definitions of *block* and *prune*.

Proof of $\Rightarrow$ direction:

For the other way, assuming R to be universal, there is a solution preserving reduction say f, of R_{STCONN} to R. $build_R$ can then simply be defined as $build_R(1^n) = f(\langle G, 1, n\rangle)$.

[7]How is this done? n is easy to determine, and s and t labels of G are kept in store. These two will be mapped to 1 and n respectively. A vertex gets the label m, if it is the mth vertex of G (respecting, of course, the committed labellings to s and t.) Here, we assume that the input is given in the usual way: first, the list of vertices, and then the list of edges, then s and t. Any other reasonable encoding (for example, without explicit vertices list, but with s and t, and followed by the list of edges) will also serve equally well.

[8]We recall that the composition of a constant number of logspace transducers can be carried out by a single logspace transducer.

Now, let us see how we can define $prune_R(\langle x, \alpha, S\rangle)$. Let g be a solution preserving reduction of R to R_{STCONN}, and assume $g(x)$ to be $\langle z_1, \gamma\rangle$. Let $prune_{R_{\text{STCONN}}}(\langle z_1, \alpha\cdot\gamma, S\rangle)$ be $\langle z_2, \delta\rangle$. (It is clear that the composed block-mask $\alpha \cdot \gamma$ will be admissible for blocks of x solutions). Next, let $f(z_2)$ be $\langle x_1, \epsilon\rangle$. Then, $prune(\langle x, \alpha, S\rangle)$ will be $\langle x_1, \delta \cdot \epsilon\rangle$.

The reason this definition of $prune_R$ works is as follows. Clearly, the appropriately pruned version of the set of solutions of x that we need is $proj_\delta(sol_{R_{\text{STCONN}}}(z_2))$. However, we have that $sol_{R_{\text{STCONN}}}(z_2) = proj_\epsilon(sol_R(x_1))$. Therefore, $proj_\delta(proj_\epsilon(sol_R(x_1))$ is the value of $prune_R$ on the specified argument.

■

4 Universality of directed graph $s - t$ unconnectivity

4.1 NL relation for $s - t$ unconnectivity

From the Immerman-Szelepscényi result [I88], we know that the complement of directed graph connectivity problem is NL-complete. We argue in this section that the NL-relation, which follows from the inductive counting technique as used in the proof of the Immerman-Szelepscényi result, does possess both *build* and *prune* operations, and therefore, the relation is NL-universal. We need the following result to state precisely what this relation is.

Theorem 16 *Let G be a directed graph with n vertices, and with two distinguished vertices s and t. Then, we can construct in logspace another directed graph G' with $O(n^8)$ vertices and with distinguished vertices s' and t' such that there is* no *directed path from s to t in G iff there is a directed path from s' to t' in G'.*

Terminology: *We will refer to the graph G' as the Immerman-Szelepcsényi graph of G, where G and G' are as in the Theorem 16 above.*

We define below an NL relation for the graph *s-t unconnectivity*, we denote this NL relation as R_{STUNCONN}.

Definition 17 ($R_{\textbf{STUNCONN}}$) *Our NL relation for the graph* s-t unconnectivity*, $R_{STUNCONN}$, will relate $\langle G, s, t\rangle$ to y if*

1. *The vertex s is not connected to vertex t in the directed graph G, and*
2. *y is an $s' - t'$ path in G' where G' is the Immerman-Szelepcsényi graph of G (possibly with some nodes in the path occurring several times consecutively in the path*

y to satisfy the constraint that all solutions of instances of same size to have the same length. To render this possible, we will have a self-loop at each node of the Immerman-Szelepcsényi graph).

4.2 Construction of Immerman-Szelepcsényi graph

We need some details about Immerman-Szelepcsényi graph to argue that R_{STUNCONN} has both *build* and *prune* operations.

The construction of Immerman-Szelepcsényi graph is based directly on the proof that nondeterministic space classes are closed under complementation [I88]. Consider a graph G with n vertices, denoted by labels 1 to n. For a nondeterministic machine M to conclude that n is not reachable from 1 in the graph G, first M computes all N_ds, the number of vertices reachable from 1 in d or less steps, with d varying from 1 to n. Computation of N_ds is done inductively. Then, M guesses N_n distinct verices, verifies that each is reachable from 1, and is different from n. At this point M concludes that n is not reachable from 1 in G, and goes into its accepting state.

M uses the following nondeterministic algorithm to obtain N_{d+1} once N_d is computed. It is assumed, without loss of generality, that there is a self-loop on every vertex of the graph G. (This is helpful to ensure every solution to have the same length).

Code fragment that computes N_{d+1} given N_d.
Here, the vertex set of input graph is $\{1,\dots,n\}$, with 1 as the source vertex and n as the target vertex.

```
$N_{d+1} = 0$;
for $i = 1$ to $n${
/* this loop determines if $i$ contributes to $N_{d+1}$ */
   check-count = 0; flag = 0;
   for j = 1 to n {
   /* this loop determines if $j$ contributes to $N_d$ */
      nondeterministically decide if vertex j is reachable from 1
      in d or less steps
      if decision is yes {
         k = 1;
         for l = 1 to d {
            nondeterministically choose an edge $(k, k')$;
            k = k';
         }
         if (k $\neq$j) reject;
         else { check-count++;
            if( i is equal to j or $(i, j)$ is an edge)
```

```
                flag = 1;
            }
        }
    }
    if (check-count $\neq N_d$) reject;
    else if (flag is equal to 1) $N_{d+1}$++;
}
/* at this point the computation of $N_{d+1}$ is complete */
```

In the overall algorithm that M uses, first, the above is iterated n times successively to compute all N_d's, $1 \leq d \leq n$. This completes what can be termed as the first phase of the algorithm. In the second phase, the algorithm essentially checks if n is among the N_n vertices reachable from source vertex 1. Verifying that vertex n is not in this set of vertices, M accepts. Phase 2 algorithm too can be made very similar to the code fragment given above: in this phase, when a path is found from 1 to j then `flag` is set if i is equal to j, and not when i is a neighbour of j as done in Phase 1.

As we said, the definition of Immerman-Szelepcsényi graph follows M's algorithm, which is basically several iterations of the code fragment shown above, with some minor variations. A node of Immerman-Szelepcsényi graph can be seen as a record with several fields (Figure 1) except for two special nodes, START and ACCEPT, we treat START as the source vertex and ACCEPT as the target vertex. ACCEPT will be reachable from START iff n in G is not reachable from 1.

In a node of Immerman-Szelepcsényi graph, the field PHASE will have value 1 when the algorithm is in its first phase. For the next phase, PHASE value will be 2. All other fields as shown in Figure 1 correspond to the variables as shown in the code fragment.

PHASE	d	N_d	N_{d+1}	i	j	k	l	flag	check-count

Figure 1: Node of a Immerman-Szelepcsényi graph

From the algorithm, along with the definition of G, it is fairly straight-forward to define what the edges of Immerman-Szelepcsényi graph will be. For example, a node w with PHASE value 1, and with l value less than d, will have an edge to node w', w' has all values same as that in w, except that l value will be one more, k field in w' will have value k', *provided* (k, k') is an edge in G. The detailed definition of Immerman-Szelepcsényi graph for G can be found in [CS04].

Even from the above condensed description of Immerman-Szelepcsényi graph for a graph G, it is not difficult to be convinced that

Proposition 18 *If there is a path p in G from 1 to n, then there will be a path p' in Immerman-Szelepcsényi graph such that the path p can be recovered from p' using a suitable block-mask.*

4.3 Existence of *build* and *prune* for R_{STUNCONN}

Suppose G is a complete directed graph on the vertex set $\{1, \ldots, n\}$. Let G_1 be the Immerman-Szelepcsényi graph of G, and let G_2 be the Immerman-Szelepcsényi graph of G_1. Note that as n is reachable from 1 in G, ACCEPT of G_1 is not reachable from START of G_1, this in turn means that ACCEPT of G_2 is reachable from START of G_2. Further, a path from START to ACCEPT in G_2 is a R_{STUNCONN} solution witnessing that in G_1 ACCEPT is *not* reachable from START. Now, making appeals to Proposition 18, we can conclude that an $1-n$ path in G can be retrieved from a R_{STUNCONN} solution of G_1. As G has all possible paths from 1 to n, a suitable blocked projection of the set of R_{STUNCONN} solutions of G_1 will span through all these paths. We see, therefore, *build* for R_{STUNCONN} on input 1^n should simply output G_1, along with a suitable block-mask.

Similarly, it is not difficult to see that R_{STUNCONN} has *prune*. Input to *prune* is a $G, 1, n$, a block-mask α, and a set S of tuples of strings. *prune* needs to prune out all R_{STUNCONN} solutions of $G, 1, n$ which, when projected with α, will have an occurrence of ss', for some $(s, s') \in S$. Any R_{STUNCONN} solution of $\langle G, 1, n \rangle$ will be a path from START to ACCEPT in the Immerman-Szelepcsényi graph G' of G. Consider an logspace tarnsducer T which is (conceptually) the composition two logspace machines T_1 and T_2. The first machine T_1 computes the Immerman-Szelepcsényi graph G' of G. What T_2 does is very similar to how we defined prune_R in the $\Rightarrow$ part of Theorem 15. T_2 considers each edge of G' one after another, and determines if this edge's occurrence in a path will lead to obtaining ss', for any $(s, s') \in S$, when the path is projected through α. If so, it determines the necessary update of the edge set of G that is required. We again appeal to Proposition 18 to argue that T_2 is can indeed determine the necessary update. Output of T is G with all updates done, along with a suitable block-mask.

5 Concluding remarks

We believe that the present work is another evidence that the notion of universality is fairly robust: for language classes defined through relations, it is indeed possible to define universality for the corresponding class of relations, and provide its structural characterization. It is not clear to us whether or not a better structural characterization of NL universality is possible than what we have provided here. In case of NP universality, one can considerably simplify many completeness proofs making use of the structural characterization. On the face of it, this does not seem to be the case with our structural characterization of NL universality. Therefore, it may be worthwhile to look for a better structural characterization of NL universality. One of our original motivation to define universality was to understand what natural reductions preserve beyond membership. The approach that has been taken so far in answering this question (through NP, #P, or NL universality) will clearly not work for some classes of reductions, in particular, for randomized reductions. Can we answer the question satisfactorily for such classes of reductions as well? This appears to us an issue worth investigation.

References

[AB92] Manindra Agrawal and Somenath Biswas, Universal Relations. In *Proc. 7th Structure in Complexity Theory Conference,* pp 207–220, 1992.

[BKT94] Harry Buhrman, Jim Kadin, and Thomas Thierauf, On Functions Computable with Nonadaptive Queries to NP. In *Proc. 9th Structure in Complexity Theory Conference,* pp 43–52, 1994.

[I88] N. Immerman, Nondeterministic space is closed under complementation, SIAM Jl. of Computing, pp 935–938, 1988.

[JLL76] N.D. Jones, E. Lien, and W.T. Lasser, New problems complete for nondeterministic logspace, Math. Syst. Theory, 10, 1, pp 1–17.

[P98] Natacha Portier, Réesolutions universelles pour des problèmes NP-complets, Theoretical Computer Science (201) 1–2, pp 137–150, 1998.

[CK03] BTP report of Chakraborty and Kumar available at `http://www.cse.iitk.ac.in/reports`

[FM03] Fournier and Malod report

[CS04] BTP report of Chaudhary abd Sinha available at `http://www.cse.iitk.ac.in/reports`