Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 158 (2010)

Realizable Paths and the NL vs L Problem

Shiva Kintali

School of Computer Science,
Georgia Institute of Technology,
Atlanta, GA 30332-0765.
kintali@cc.gatech.edu

Abstract

A celebrated theorem of Savitch [Sav70] states that $NSPACE(S) \subseteq DSPACE(S^2)$. In particular, Savitch gave a deterministic algorithm to solve ST-CONNECTIVITY (an $\mathbf{NL}$-complete problem) using $O(\log^2 n)$ space, implying $\mathbf{NL} \subseteq DSPACE(\log^2 n)$. While Savitch's theorem itself has not been improved in the last four decades, several graph connectivity problems are shown to lie between $\mathbf{L}$ and $\mathbf{NL}$, providing new insights into the space-bounded complexity classes. All the connectivity problems considered in the literature so far are essentially special cases of ST-CONNECTIVITY.

In the first half of this paper, we initiate the study of auxiliary PDAs as graph connectivity problems and define sixteen different *graph realizability problems* and study their relationships. The complexity of these connectivity problems lie between $\mathbf{L}$ and $\mathbf{P}$. ST-REALIZABILITY, the most general graph realizability problem is $\mathbf{P}$-complete. 1DSTREAL(POLY), the most specific graph realizability problem is $\mathbf{L}$-complete. As special cases of our graph realizability problems we define two natural problems, BALANCED ST-CONNECTIVITY and POSITIVE BALANCED ST-CONNECTIVITY, that lie between $\mathbf{L}$ and $\mathbf{NL}$.

In the second half of this paper, we study the space complexity of $\mathbf{SGSLogCFL}$, a graph realizability problem lying between $\mathbf{L}$ and $\mathbf{LogCFL}$. We define generalizations of graph squaring and transitive closure, present efficient parallel algorithms for $\mathbf{SGSLogCFL}$ and use the techniques of Trifonov [Tri08] to show that $\mathbf{SGSLogCFL}$ is contained in $DSPACE(\log n \log\log n)$. This implies that BALANCED ST-CONNECTIVITY is contained in $DSPACE(\log n \log\log n)$. We conclude with several interesting new research directions.

Keywords: auxiliary pushdown automata, graph connectivity problems, $\mathbf{LogCFL}$, parallel graph algorithms, Savitch's theorem, space-bounded computation, st-connectivity, symmetric Turing machines.

ISSN 1433-8092

1 Introduction

A celebrated theorem of Savitch [Sav70] states that $NSPACE(S) \subseteq DSPACE(S^2)$. In particular, Savitch gave a deterministic algorithm to solve ST-CONNECTIVITY (an $\mathbf{NL}$-complete problem) using $O(\log^2 n)$ space, implying $\mathbf{NL} \subseteq DSPACE(\log^2 n)$. ST-CONNECTIVITY (in short STCONN) is the problem of determining whether there exists a path between two distinguished vertices s and t in a directed graph. Savitch's algorithm runs in time $2^{O(\log^2 n)}$. It is a longstanding open problem to improve Savitch's theorem i.e., to prove (i) $\mathbf{NL} \subseteq DSPACE(o(\log^2 n))$ or (ii) $\mathbf{NL} \subseteq \mathbf{SC^2}$, i.e., STCONN can be solved by a deterministic algorithm in polynomial time and $O(\log^2 n)$ space.

While Savitch's theorem itself has not been improved in the last four decades, several graph connectivity problems are shown to lie between $\mathbf{L}$ and $\mathbf{NL}$, providing new insights into the space-bounded complexity classes. Allender's survey [All07] gives an update of progress related to several special cases of STCONN. Recently STCONN in planar DAGs with $O(\log n)$ sources is shown to be in $\mathbf{L}$ [SBV10]. Stolee and Vinodchandran proved that STCONN in DAGs with $2^{O(\sqrt{\log n})}$ sources embedded on surfaces of genus $2^{O(\sqrt{\log n})}$ is in $\mathbf{L}$ [SV10].

All the connectivity problems considered in the literature so far are essentially special cases of STCONN. In the first half of this paper, we initiate the study of auxiliary PDAs as graph connectivity problems and define sixteen different *graph realizability problems* and study their relationships. The complexity of these connectivity problems lie between $\mathbf{L}$ and $\mathbf{P}$. STREAL, the most general graph realizability problem is $\mathbf{P}$-complete. 1DSTREAL(POLY), the most specific graph realizability problem is $\mathbf{L}$-complete. As special cases of our graph realizability problems we define two natural problems, BALANCED ST-CONNECTIVITY and POSITIVE BALANCED ST-CONNECTIVITY, that lie between $\mathbf{L}$ and $\mathbf{NL}$.

In the second half of this paper, we study the space complexity of $\mathbf{SGSLogCFL}$ (See Section 4 for definition). We define generalizations of graph squaring and transitive closure, present efficient parallel algorithms for $\mathbf{SGSLogCFL}$ and use the techniques of Trifonov [Tri08] to show that $\mathbf{SGSLogCFL}$ is contained in $DSPACE(\log n \log\log n)$. This implies that BALANCED ST-CONNECTIVITY is contained in $DSPACE(\log n \log\log n)$.

1.1 Preliminaries, Related Work and Our Results

Auxiliary Pushdown Automata : A language is accepted by a non-deterministic pushdown automaton (PDA) if and only if it is a context-free language. Deterministic context-free languages are those accepted by the deterministic PDAs. $\mathbf{LogCFL}$ is the set of all languages that are log-space reducible to a context-free language. Similarly, $\mathbf{LogDCFL}$ is the set of all languages that are log-space reducible to a deterministic context-free language. There are many equivalent characterizations of $\mathbf{LogCFL}$. Sudborough [Sud78] gave the machine class equivalence. Ruzzo [Ruz80] gave an alternating Turing machine (ATM) class equivalent to $\mathbf{LogCFL}$. Venkateswaran [Ven91] gave a circuit characterization and showed that $\mathbf{LogCFL} = \mathbf{SAC^1}$. For a survey of parallel complexity classes and $\mathbf{LogCFL}$ see Limaye's thesis [Lim05].

An Auxiliary Pushdown Automaton (NAuxPDA or simply AuxPDA), introduced by Cook [Coo71], is a two-way PDA augmented with an $S(n)$-space bounded work tape. If a deterministic two-way PDA is augmented with an $S(n)$-space bounded work tape then we get a Deterministic Auxiliary Pushdown Automaton (DAuxPDA). We present the formal definitions in the **appendix** (see Section A). Let *NAuxPDA-SpaceTime* $(S(n),T(n))$ be the class of languages accepted by an AuxPDA with $S(n)$-space bounded work tapes and the running time bounded by $T(n)$. Let the corresponding deterministic class be *DAuxPDA-SpaceTime* $(S(n),T(n))$. It is easy to see that $\mathbf{NL} \subseteq$ *NAuxPDA-SpaceTime* $(O(\log n), poly(n))$. It is shown by Sudborough that *NAuxPDA-SpaceTime* $(O(\log n), poly(n))$ = $\mathbf{LogCFL}$ and *DAuxPDA-SpaceTime* $(O(\log n),poly(n))$

= $\mathbf{LogDCFL}$ [Sud78]. Using ATM simulations, Ruzzo showed that $\mathbf{LogCFL} \subseteq \mathbf{NC^2}$ [Ruz80]. Simpler proofs of *DAuxPDA-SpaceTime* $(O(\log n), poly(n))$ = $\mathbf{LogDCFL}$ and $\mathbf{LogCFL}$ = $\mathbf{SAC^1}$ are given in [MRV99].

Many proof techniques and results obtained in the context of **NL**, are generalized to obtain the corresponding results for $\mathbf{LogCFL}$. For example : (i) Borodin [Bor77] proved that $\mathbf{NL} \subseteq \mathbf{NC^2}$. Ruzzo [Ruz80] introduced tree-size-bounded alternating Turing machines, gave a new characterization of $\mathbf{LogCFL}$, and proved that $\mathbf{LogCFL} \subseteq \mathbf{NC^2}$. (ii) Immerman [Imm88] and Szelepcsényi [Sze87] proved that **NL** = **co-NL**. Borodin et. al. [BCD$^+$89] generalized their inductive counting technique and proved that $\mathbf{LogCFL}$ = **co-**$\mathbf{LogCFL}$. In fact, they proved a stronger result showing that $\mathbf{SAC}^i$ is closed under complementation for $i > 0$. (iii) Wigderson [Wig94] proved that $\mathbf{NL} \leq_r \oplus\mathbf{NL}$. Gál and Wigderson [GW96] proved that $\mathbf{LogCFL} \leq_r \oplus\mathbf{LogCFL}$. (iv) Nisan [Nis94] proved that $\mathbf{BPL} \subseteq \mathbf{SC^2}$. Venkateswaran [Ven06, Ven09] proved that $\mathbf{BPLogCFL} \subseteq \mathbf{SC^2}$ and $\mathbf{BPLogCFL} \subseteq \mathbf{NC^2}$. Here $\mathbf{BPLogCFL}$ (resp. $\mathbf{RLogCFL}$ and $\mathbf{ZPLogCFL}$) is the bounded error (resp. one-sided error and zero error) probabilistic version of $\mathbf{LogCFL}$. All the above results are elegant and non-trivial generalizations of the corresponding results in the logspace setting.

Throughout this paper, we consider $O(\log n)$-space bounded AuxPDAs. The *surface configuration* (introduced by Cook [Coo71]) of an AuxPDA, on an input w, consists of the state, contents and head positions of the work tapes, the head position of the input tape and the topmost symbol of the stack i.e., the rightmost symbol of the pushdown tape. Note that for an $S(n)$-space bounded AuxPDA, its surface configurations take only $O(S(n))$ space. In the rest of the paper, we will refer to surface configurations as configurations. For an input w, a pair of configurations (C_1, C_2) is *realizable* if the AuxPDA can move from C_1 to C_2 ending with its stack at the same height as in C_1, and without popping its stack below its level in C_2 for any of the intermediate configurations. An AuxPDA M accepts an input w iff there is a realizable pair (I, A), where I is the initial configuration and A is the unique accepting configuration.

ST-REALIZABILITY : In Section 2, we initiate the study of auxiliary PDAs as graph connectivity problems and define STREAL and several special cases of STREAL. Our definition of STREAL is motivated by (i) Hardest CFL [Gre73, Sud78, Har78], (ii) Labeled Acyclic GAP, which is $\mathbf{LogCFL}$-complete [GHR95] (iii) CFL-reachability, which is $\mathbf{P}$-complete [MR00, AP87, Rep96, UG86] and (iv) the insights from Niedermeier and Rossmanith's parsimonious simulation of $\mathbf{LogCFL}$ by $\mathbf{SAC^1}$ circuits [NR95].

Symmetric AuxPDAs : In Section 2.2, we define USTREAL, a symmetric version of STREAL. To study the space complexity of USTREAL we define *symmetric* auxiliary pushdown automata, a natural generalization of symmetric Turing machines introduced by Lewis and Papadimitriou [LP82]. We introduce a new complexity class called $\mathbf{SLogCFL}$ (a generalization of **SL**) and show that $\mathbf{LogDCFL} \subseteq \mathbf{SLogCFL} \subseteq \mathbf{LogCFL}$.

More Graph Realizability Problems : In Sections 5 and 6, we study several variants of STREAL and the corresponding complexity classes. We study the relationship between sixteen different *graph realizability problems*, whose complexity lies between **L** and **P**. BALANCED ST-CONNECTIVITY and POSITIVE BALANCED ST-CONNECTIVITY are two natural graph connectivity problems that lie between **L** and **NL**. Figure 1 summarizes the relationship among the newly defined classes.

Generalizations of Transitive Closure and Graph Squaring : Unlike STCONN, using breadth-first search (or) depth-first search and keeping track of "visited" vertices does not result in efficient algorithms for STREAL. In Section 7, we generalize the notions of transitive closure and graph squaring. Using these generalizations we present a natural repeated squaring algorithm to compute the generalized transitive closure, thus solving STREAL.

Space Efficient Algorithms : STCONN (resp. USTCONN) is the problem of determining whether there exists a path between two distinguished vertices s and t in a directed (resp. undirected) graph. These two graph connectivity problems played a central role in understanding the complexity classes $\mathbf{L}$, $\mathbf{SL}$ and $\mathbf{NL}$ [AKL$^+$79, LP82, BCD$^+$89, NSW92, KW93, NT95, SZ99, ATWZ00, RVW00, Tri08, Rei08].

The $\mathbf{L}$ vs $\mathbf{SL}$ question (i.e., is there a log space algorithm for solving USTCONN) motivated an exciting series of new concepts and techniques. Lewis and Papadimitriou [LP82] introduced symmetric Turing machines to study the space complexity of USTCONN. Prior to their work, Aleliunas et. al. [AKL$^+$79] proved that USTCONN $\in \mathbf{RL}$, implying $\mathbf{SL} \subseteq \mathbf{RL}$. Nisan, Szemeredi and Wigderson [NSW92] showed that USTCONN can be solved deterministically in space $O(\log^{\frac{3}{2}} n)$. This result was later subsumed by a beautiful result of Saks and Zhou, showing that $BP_H SPACE(S) \subseteq DSPACE(S^{3/2})$ [SZ99]. Armoni, et. al. [ATWZ00] showed that USTCONN $\in DSPACE(\log^{\frac{4}{3}} n)$. Trifonov [Tri08] gave an $O(\log n \log\log n)$-space deterministic algorithm for USTCONN. Independently at the same time, using completely different techniques, Reingold [Rei08] settled the space complexity of USTCONN and proved that $\mathbf{SL} = \mathbf{L}$. The zig-zag graph product, introduced by Reingold, Vadhan and Wigderson [RVW02], played a crucial role in Reingold's algorithm. Rozenman and Vadhan [RV05] introduced a derandomized analogue of graph squaring and presented an alternative proof of Reingold's theorem.

One of our goals in this paper is to develop techniques to design space efficient algorithms for graph realizability problems. In particular, we study the space complexity of $\mathbf{SGSLogCFL}$. Applying the techniques of [RVW02, Rei08, RV05] to design space-efficient algorithms for $\mathbf{SGSLogCFL}$ seems to be a challenging task. Our space efficient algorithm for $\mathbf{SGSLogCFL}$ (see Section 9) is based on Trifinov's technique [Tri08]. The main idea in [Tri08] is to space-efficiently simulate parallel algorithms for USTCONN. Trifonov's proof of $\mathbf{SL} \subseteq DSPACE(\log n \log\log n)$ is based on Chong-Lam's parallel algorithm [CL95] solving USTCONN in $O(\log n \log\log n)$ time on EREW PRAM.

Research in parallel algorithms for USTCONN has a rich history. Hirschberg, Chandra and Sarwate [HCS79] presented an $O(\log^2 n)$ time parallel algorithm using $n^2/\log n$ processors on a CREW PRAM to find connected components of an undirected graph. Their algorithm remained the best known for almost a decade. In a breakthrough work, Johnson and Metaxas [JM97] presented a CREW algorithm running in $O(\log^{\frac{3}{2}} n)$ time using $n + m$ processors. Subsequently they improved their algorithm to run on an EREW PRAM with the same time complexity and number of processors [JM95]. Chong and Lam [CL95] presented an $O(\log n \log\log n)$ time deterministic EREW PRAM algorithm with $O(m+n)$ processors. Chong, Han, and Lam [CHL99] showed that the problem can be solved on the EREW PRAM in $O(\log n)$ time with $O(m+n)$ processors.

In Section 8, we start by generalizing the parallel algorithm of [HCS79]. This generalization introduces the basic connections between our generalized graph squaring and the *hook and contract* based parallel algorithms. These connections play a crucial role in understanding the parallel algorithms in the subsequent sections. We then generalize the algorithms of [JM97] and [CL95] and design the corresponding parallel algorithms for $\mathbf{SGSLogCFL}$. In Section 9, we use these generalizations and the techniques of Trifonov [Tri08] to prove that $\mathbf{SGSLogCFL}$ is contained in $DSPACE(\log n \log\log n)$. This implies that BALANCED ST-CONNECTIVITY is contained in $DSPACE(\log n \log\log n)$.

1.2 Why is Balanced ST-Connectivity an interesting and important problem ?

It is well known that STCONN is $\mathbf{NL}$-complete. Savitch gave a deterministic algorithm to solve STCONN, implying $\mathbf{NL} \subseteq DSPACE(\log^2 n)$. Reingold proved that **SL** = **L**, thus showing that USTCONN is $\mathbf{L}$-complete. The zig-zag graph product, introduced by Reingold, Vadhan and Wigderson [RVW02], played a crucial role in Reingold's algorithm. Unfortunately it is also known that these techniques are not applicable to STCONN. Reingold's algorithm heavily relies on the "undirected" nature of USTCONN and the well-studied notion of (undirected) expander graphs. While there are notions of *directed* graph expansion, they do not seem to be helpful in generalizing Reingold's techniques to improve Savitch's theorem. The "directed" nature of STCONN is frustrating all attempts of applying the techniques of [RVW02, Rei08, RV05] to improve its space complexity. Are there intermediate problems between $\mathbf{L}$ and $\mathbf{NL}$ that can help us better understand the limitations of these techniques ?

Prior to our work all the connectivity problems between $\mathbf{L}$ and $\mathbf{NL}$ are essentially special cases of STCONN. Balanced ST-Connectivity is a new kind of a natural graph connectivity problem (see Section 6 for definition) lying between $\mathbf{L}$ and $\mathbf{NL}$. An instance of Balanced ST-Connectivity is a *directed graph*. We show that this instance can be represented using two *symmetric* matrices, a standard matrix and a gap matrix (see Section 2.5 for definitions). The symmetry of these matrices combined with our generalized graph squaring procedure (see Section 7.2) allowed us to naturally generalize the known parallel algorithms of USTCONN, which in turn allowed us to apply Trifonov's techniques and prove that Balanced ST-Connectivity $\in DSPACE(\log n \log\log n)$ (see Corollary 9.2). The main insight from this result is that Trifonov's techniques are applicable in a much more general setting than just USTCONN. On the other hand there does not seem to be a natural way of applying the techniques of [RVW02, Rei08, RV05] to Balanced ST-Connectivity and achieve even $o(\log^2 n)$ upper bound.

On one hand, our work motivates further investigation of applicability of Trifonov's techniques in more general settings. On the other hand, our work leaves a challenging task of generalizing Reingold's techniques to Balanced ST-Connectivity. After all, Balanced ST-Connectivity has lots of "symmetry" in its definition. What concepts are we missing, to apply Reingold's techniques to Balanced ST-Connectivity ? Do we need an appropriate generalization of *graph expansion* defined in the context of Balanced ST-Connectivity ? Do we need a new kind of graph product (along the lines of zig-zag graph product and replacement product) ? Investigating such expansion parameters and graph products is an interesting research direction towards understanding and developing the right notion of "directed expansion" and graph products to be used in the context of STCONN.

As mentioned earlier, $\mathbf{SGSLogCFL}$ is a generalization of Balanced ST-Connectivity. One of the most interesting open problems arising from our work is to improve Theorem 9.1. Is $\mathbf{SGSLogCFL} \in \mathbf{L}$? Resolving this seems to be a challenging task. An intermediate step is to prove that $\mathbf{SGSLogCFL} \in \mathbf{SC^2}$. One way to achieve this goal is to prove that $\mathbf{SGSLogCFL} \in \mathbf{RL}$ and use Nisan's theorem ($\mathbf{RL} \subseteq \mathbf{SC^2}$) [Nis94].

One of the oldest results concerning the space complexity of USTCONN is that of Aleliunas et. al. [AKL$^+$79]. They proved that USTCONN $\in \mathbf{RL}$. Their algorithm is very elegant. Simply start from the source node (say s) and perform a random walk for polynomial number of steps. If the destination node (say t) is visited within these steps, declare that s and t are connected. Can we generalize their techniques to prove that $\mathbf{SGSLogCFL} \in \mathbf{RL}$? This approach necessitates an appropriate generalization of *random walks* and associated concepts like *cover time*. Defining such concepts will shed new light on the connectivity problems that are intermediate between $\mathbf{L}$ and $\mathbf{NL}$.

Our work opens up several such intriguing new research directions. They are discussed in Section 10. To improve Savitch's theorem it is important that we seek answers to these questions.

2 Realizable Paths

As mentioned earlier, an *auxiliary pushdown automaton* (AuxPDA) is a multi-tape Turing machine with a two-way read-only input tape, a pushdown tape, and one or more work tapes. The pushdown alphabet has a distinguished symbol (say \$) which is initially *pushed* on the pushdown tape. The pushdown head never shifts left of \$ or changes \$. Further, the pushdown head can never shift left when scanning any tape symbol unless it first erases (i.e., pops) that symbol, and it can never shift right from a square unless it first prints (i.e., pushes) a nonblank symbol on that square. If the Turing machine is non-deterministic we get a non-deterministic Auxiliary Pushdown Automaton (NAuxPDA or simply AuxPDA). If the Turing machine is deterministic we get a deterministic Auxiliary Pushdown Automaton (DAuxPDA). *Space on an AuxPDA* is the space used on the work tapes without counting the space on the pushdown tape. In this paper we use the terms pushdown tape and stack interchangeably.

Throughout this paper, we consider AuxPDAs that are $O(\log n)$-space bounded. We will assume that an AuxPDA (i) accepts with its stack empty and halts on all computations, and (ii) there is a unique accepting configuration (assuming an empty pushdown tape, an empty work tape, a fixed position for all tape heads, and a unique final accepting state).

In this section, we first define a graph connectivity problem called ST-REALIZABILITY (in short STREAL) that captures the computations of an AuxPDA. We are given a directed graph (say $\mathcal{G}$) where each vertex has a *stack symbol* associated with it and each edge is labeled $push$, *pop* or ϵ. Let s and t be two distinguished vertices of $\mathcal{G}$ and let P be a directed (not necessarily simple) path from s to t. We maintain a stack and traverse the path P starting from s by first pushing the label of s into the stack. Throughout the traversal the top of the stack is called the *current symbol*. We traverse the edges of P according to the following rules :

- We are allowed to traverse along any edge $e = (u, v)$ labeled ϵ, without modifying the stack, provided the label of v is same as the current symbol.
- While traversing any edge $e = (u, v)$ labeled $push$, we have to push the label of v into the stack, so that it becomes the current symbol.
- We are allowed to traverse any edge $e = (u, v)$ labeled *pop*, so long as the label associated with the vertex v agrees with what would become the current symbol after popping the top symbol off the stack.

If we reach the vertex t with the stack in the same configuration as we started at s, then the path P is called a *realizable* path. This traversal defines a computation path of an AuxPDA and each vertex of $\mathcal{G}$ represents a *surface configuration* of the AuxPDA. The graph $\mathcal{G}$ represents the configuration graph of the AuxPDA. The terms *realizable paths* and *surface configuration* are first introduced by Cook [Coo71]. We now present a formal definition of STREAL.

2.1 STREAL

We are given a directed graph $\mathcal{G}(V, E)$, a vertex labeling function $L_V : V \to \{\alpha_1, \alpha_2, \dots, \alpha_k\}$ and an edge labeling function $L_E : E \to \{push, pop, \epsilon\}$. The ordered pair (s, t), where $s, t \in V$, is said to be **realizable** if the following two conditions hold :

- There is a directed path (say P) from s to t.
- The concatenation of the vertex and edge labels along the path P is a *realizable* string (see Definition 2.1).

Definition 2.1. Let $\mathcal{A} = \{push, pop, \epsilon, \alpha_1, \alpha_2, \ldots, \alpha_k\}$ be the set of alphabets. A **realizable string** is a nonempty string of alphabets from $\mathcal{A}$, defined in the following recursive manner :

- for all $1 \leq i \leq k$, "α_i" is a realizable string.
- for all $1 \leq i \leq k$, "$\alpha_i\ \epsilon\ \alpha_i$" is a realizable string.
- if S is a realizable string then so is "$\alpha_i\ push\ S\ pop\ \alpha_i$", for all $1 \leq i \leq k$.
- for all $1 \leq i \leq k$, if "$\alpha_i\ S_1\ \alpha_i$" and "$\alpha_i\ S_2\ \alpha_i$" are realizable strings then so is "$\alpha_i\ S_1\ \alpha_i\ S_2\ \alpha_i$".

STREAL : Given a directed graph $\mathcal{G}(V, E)$ with vertices labeled from $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ and edges labeled from $\{push, pop, \epsilon\}$ and two distinguished nodes s and t, decide if there is a realizable path from s to t in $\mathcal{G}$.

We use the notation $(u \rightsquigarrow v)$ to denote that there is a realizable path from u to v. If all the vertices of $\mathcal{G}$ are labeled α_1 (i.e., $k = 1$) and all the edges are labeled ϵ, we get an instance of STCONN. Hence, STREAL is a generalization of STCONN.

2.2 USTREAL

We now define USTREAL, a symmetric version of STREAL. USTREAL captures the computation of *symmetric* AuxPDAs. Intuitively, a *symmetric* AuxPDA is a nondeterministic multi-tape Turing machine which has an extra tape called pushdown tape, with an additional requirement that every move of the machine is "reversible". In other words, the "yields" relation between its (surface) configurations is symmetric. Such a machine is allowed to scan two symbols at a time on each of its tapes. We present the formal definitions and properties of symmetric AuxPDAs in the appendix (see appendix A).

We are given an undirected graph $\mathcal{G}(V, E)$, a vertex labeling function $L_V : V \rightarrow \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ and an edge labeling function $L_E : E \rightarrow \{push, pop, \epsilon\}$. Moreover, the edge labels are "symmetric" i.e., they satisfy the following properties : (i) $L_E(u, v) = push$ if and only if $L_E(v, u) = pop$ and (ii) $L_E(u, v) = \epsilon$ if and only if $L_E(v, u) = \epsilon$.

The pair (s, t), where $s, t \in V$, is said to be *realizable* if there is an undirected path (say P) from s to t and the concatenation of the vertex and edge labels along the path P is a *realizable* string. Since the edge labels are symmetric, (s, t) is realizable if and only if (t, s) is realizable. We denote this by $(s \leftrightsquigarrow t)$.

USTREAL : Given an undirected graph $\mathcal{G}(V, E)$ with vertices labeled from $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ and *symmetric* edge labels from $\{push, pop, \epsilon\}$ and two distinguished nodes s and t, decide if s and t are realizable in $\mathcal{G}$.

If all the vertices of $\mathcal{G}$ are labeled α_1 (i.e., $k = 1$) and all the edges are labeled ϵ, we get an instance of USTCONN. Hence, USTREAL is a generalization of USTCONN.

2.3 DSTREAL

We now define a deterministic version of STREAL, called DSTREAL, capturing the behavior of deterministic AuxPDAs. An instance of DSTREAL is a graph with vertex labels and edges labels similar to STREAL. The definition of a realizable path is also the same. The graph $\mathcal{G}$ associated with DSTREAL satisfies the following properties reflecting the deterministic behavior of computation paths of a deterministic AuxPDA.

- Let $e = (u, v)$ be a directed edge of $\mathcal{G}$. If (u, v) is labeled ϵ, then e is the *only* out-going edge from u.
- Let $e = (u, v)$ be a directed edge of $\mathcal{G}$. If (u, v) is labeled $push$, then e is the *only* out-going edge from u.
- Let u have l out-going edges all labeled *pop*. Let $v_1, v_2, \ldots, v_l$ be the out-neighbors of u. Then the labels of the vertices $v_1, v_2, \ldots, v_l$ are all distinct.

DSTREAL : Given a directed graph $\mathcal{G}(V, E)$ with vertices labeled from $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ and edges labeled from $\{push, pop, \epsilon\}$ (satisfying the above mentioned properties) and two distinguished nodes s and t, decide if there is a realizable path from s to t in $\mathcal{G}$.

2.4 Relationship between STREAL, USTREAL and DSTREAL

As mentioned earlier, AuxPDAs are introduced by Cook [Coo71]. In the same paper, Cook proved that $O(\log n)$-space bounded AuxPDAs (both deterministic and non-deterministic) accept precisely those languages that are accepted by a polynomial time bounded Turing machine. Hence, the following theorems are immediate.

Theorem 2.2. STREAL and DSTREAL are $\mathbf{P}$-complete.

We use the same names STREAL, USTREAL and DSTREAL to denote languages that are logspace reducible to the connectivity problems STREAL, USTREAL and DSTREAL respectively. Since DSTREAL $\subseteq$ USTREAL $\subseteq$ STREAL by Theorem A.2 (see appendix), we have the following corollary.

Theorem 2.3. DSTREAL = USTREAL = STREAL = $\mathbf{P}$.

2.5 Graph Representation

We now discuss the representation of an instance of STREAL i.e., a directed graph $\mathcal{G}$ with the vertex and edge labels. Let this graph be $\mathcal{G}(V, E)$ with $|V| = n$. For simplicity we assume that there are no multi-edges. We represent $\mathcal{G}$ as a 4-tuple $\mathcal{G} = \langle \mathcal{L}, \mathcal{P}_{push}, \mathcal{P}_{pop}, \mathcal{E} \rangle$, where $\mathcal{L}$ is an integer array of length n, $\mathcal{P}_{push}$, $\mathcal{P}_{pop}$ and $\mathcal{E}$ are $n \times n$ boolean matrices. $\mathcal{L}$ is an integer array of length N representing the vertex labels. $\mathcal{L}[u]$ represents the label of vertex u i.e., $\mathcal{L}[u] = i$ iff the label of u is α_i. The $[u, v]^{th}$ entry of the matrix $\mathcal{P}_{push}$ (resp. $\mathcal{P}_{pop}$ and $\mathcal{E}$) is 1 if and only if the directed edge (u, v) is labeled *push* (resp. *pop* and ϵ). We may assume that $L_E(u, u) = \epsilon$ for all $u \in V$ i.e., $\mathcal{E}[u, u] = \epsilon$ for all $u \in V$.

Definition 2.4. *(Niedermeier and Rossmanith [NR95])* : Let *a,b,c,d* be four configurations such that : *a* and *b* have same pushdown heights, *c* and *d* have same pushdown heights and there exists a computation path from *a* to *c* and one from *d* to *b*. The level of the pushdown must not go below the level of *a* and *b* during the computation. We say that *(a,b)* is *realizable with gap (c,d)*.

In the context of STREAL, we relax the above definition as shown below. This allows us to define a natural repeated squaring algorithm to solve STREAL. For the rest of this paper, we will use the following definition.

Path with gap : A *path with gap* consists of four vertices a, b, c, d such that (i) there is a computation path P_1 from a to c and P_2 from d to b (ii) the vertex labels of a and b are the same (iii) the vertex labels of c and d are the same (iv) let P be the path formed by concatenating P_1 and P_2 i.e., identifying c and d (iv) the concatenation of the vertex and edge labels along the path P is a *realizable* string. We denote such a "path with gap" by $(a \rightsquigarrow (c, d) \rightsquigarrow b)$ and say that *(a,b)* is *realizable with gap (c,d)*. The *length* of such a path with gap is the length of P.

Path with gap $(a \rightsquigarrow (c, d) \rightsquigarrow b)$ is interpreted as if the two surface configurations c and d were the same, i.e., as if a realizable path from c to d would exist. To keep track of paths with gaps, we maintain a boolean *gap matrix* Υ, indexed by 4-tuple of vertices $[a, (c, d), b]$ such that if $\Upsilon[a, (c, d), b] = 1$ then $(a \rightsquigarrow (c, d) \rightsquigarrow b)$. We initialize the gap matrix Υ with the labels from the matrices $\mathcal{L}$,$\mathcal{P}_{push}$ and $\mathcal{P}_{pop}$ as follows.

```
InitializeGapMatrix($\Upsilon$)
    for all $a, b, c, d \in V$   $\Upsilon[a, (c, d), b] = 0$
    for all $a, b, c, d \in V$
        if $((\mathcal{P}_{push}[a, c] == 1) \&\& (\mathcal{P}_{pop}[d, b] == 1) \&\& (\mathcal{L}[a] == \mathcal{L}[b]) \&\& (\mathcal{L}[c] == \mathcal{L}[d]))$
            then $\Upsilon[a, (c, d), b] = 1$
    for all $a \in V$   $\Upsilon[a, (a, a), a] = 1$
    for all $a, b \in V$   $\Upsilon[a, (a, b), b] = 1$
```

All the required information from the matrices $\mathcal{L}$,$\mathcal{P}_{push}$ and $\mathcal{P}_{pop}$ is now present in the gap matrix Υ. We call $\mathcal{E}$ the *standard* matrix and Υ the *gap* matrix and assume that an instance of STREAL, $\mathcal{H}$, is represented by an $n \times n$ standard matrix $\mathcal{E}$ and an $n^2 \times n^2$ gap matrix Υ and denote this by $\mathcal{H} = \langle \Upsilon, \mathcal{E} \rangle$. The rows and columns of Υ are indexed by pairs of vertices of $\mathcal{H}$. $\Upsilon[a, (c, d), b]$ corresponds to the $[(a, b), (c, d)]^{th}$ entry in the $n^2 \times n^2$ matrix.

3 Realizability with Symmetric Gap

As noted earlier, an instance of STREAL is represented by an $n \times n$ standard matrix $\mathcal{E}$ and an $n^2 \times n^2$ gap matrix Υ. In an instance of USTREAL, the standard matrix is symmetric. We define SGUSTREAL to be a graph realizability problem in which *both* the standard matrix and the gap matrix are symmetric. The prefix **SG** stands for symmetric gap [1]. We now give a formal definition of SGUSTREAL.

We are given an undirected graph $\mathcal{G}(V, E)$, a vertex labeling function $L_V : V \rightarrow \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ and an edge labeling function $L_E : E \rightarrow \{push, pop, \epsilon\}$. The edge labels are "symmetric" as defined in Section 2.2. The pair (s, t), where $s, t \in V$, is said to be **realizable with symmetric gap** if the following two conditions hold :

- There is an undirected path (say P) from s to t.

[1]A moment of thought would reveal that the case of symmetric gap matrix and asymmetric standard matrix does not make much sense

- The concatenation of the vertex and edge labels along the path P is a *realizable string with symmetric gap* (see Definition 3.1).

Definition 3.1. Let $\mathcal{A} = \{push, pop, \epsilon, \alpha_1, \alpha_2, \ldots, \alpha_k\}$ be the set of alphabets. A **realizable string with symmetric gap** is a nonempty string of alphabets from $\mathcal{A}$, defined in the following recursive manner :

- for all $1 \leq i \leq k$, "α_i" is a realizable string.
- for all $1 \leq i \leq k$, "$\alpha_i\ \epsilon\ \alpha_i$" is a realizable string.
- if S is a realizable string then so is "$\alpha_i\ push\ S\ pop\ \alpha_i$", for all $1 \leq i \leq k$.
- if S is a realizable string then so is "$\alpha_i\ pop\ S\ push\ \alpha_i$", for all $1 \leq i \leq k$.
- for all $1 \leq i \leq k$, if "$\alpha_i\ S_1\ \alpha_i$" and "$\alpha_i\ S_2\ \alpha_i$" are realizable strings then so is "$\alpha_i\ S_1\ \alpha_i\ S_2\ \alpha_i$".

Since the edge labels are symmetric, (s, t) is realizable if and only if (t, s) is realizable. We initialize the gap matrix as described in Section 2.5. By the definition of *realizable string with symmetric gap*, $(a \rightsquigarrow (c, d) \rightsquigarrow b)$ if and only if $(c \rightsquigarrow (a, b) \rightsquigarrow d)$. Hence the corresponding $n^2 \times n^2$ gap matrix Υ is a symmetric matrix. We denote this symmetry by $(a \leftrightsquigarrow (c, d) \leftrightsquigarrow b)$.

SGUSTREAL : Given an undirected graph $\mathcal{G}(V, E)$ with vertices labeled from $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ and *symmetric* edge labels from $\{push, pop, \epsilon\}$ and two distinguished nodes s and t, decide if s and t are *realizable with symmetric gap* in $\mathcal{G}$.

Note : STREAL, USTREAL and DSTREAL capture the behaviour of non-deterministic AuxPDAs, symmetric AuxPDAs and deterministic AuxPDAs respectively. Defining such a machine (or a circuit) characterization for SGUSTREAL is an *open problem*. In this paper, we study SGUSTREAL purely as a graph connectivity problem.

4 Polynomially bounded realizable paths

There are instances of DSTREAL (and hence USTREAL and STREAL) with a unique exponentially long realizable path from s to t [KS11]. We now define realizability problems seeking realizable paths of polynomial length. Since we are studying these connectivity problems under logspace reductions, it is sufficient to look for realizable paths of length n, where n is the number of vertices in the graph $\mathcal{G}$.

STREAL(POLY) : Given a directed graph $\mathcal{G}(V, E)$ with vertices labeled from $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ and edges labeled from $\{push, pop, \epsilon\}$ and two distinguished nodes s and t, decide if there is a realizable path of length at most $n = |V|$ from s to t in $\mathcal{G}$.

USTREAL(POLY), DSTREAL(POLY) and SGUSTREAL(POLY) are defined analogously. Sudborough showed that *NAuxPDA-SpaceTime* $(O(\log n), poly(n))$ = **LogCFL** and *DAuxPDA-SpaceTime* $(O(\log n), poly(n))$ = **LogDCFL** [Sud78]. Hence the following theorems are immediate.

Theorem 4.1. STREAL(POLY) is **LogCFL**-complete.

Theorem 4.2. DSTREAL(POLY) is **LogDCFL**-complete.

Note : It is known that the circuit class $\mathbf{SAC^i}$ is equivalent to the set of languages recognized by Nondeterministic AuxPDAs using $O(\log n)$ space and $O(\log^i n)$ stack height (for details see [Ruz80, Ven91]). We note that STREAL($n^{\log^{i-1} n}$), defined similar to STREAL(POLY), is complete for the complexity class $\mathbf{SAC^i}$.

We now define the complexity classes corresponding to USTREAL(POLY) and SGUSTREAL(POLY).

SLogCFL is the class of languages accepted by a logspace bounded and polynomial time bounded symmetric AuxPDA. In other words, **SLogCFL** is the class of languages that are logspace reducible to USTREAL(POLY).

SGSLogCFL is the class of languages that are logspace reducible to SGUSTREAL(POLY).

Independent to our work, Allender and Lange [AL10] defined symmetric AuxPDAs and proved that every language accepted by a nondeterministic auxiliary pushdown automaton in polynomial time can be accepted by a symmetric auxiliary pushdown automaton in polynomial time. Their definition of symmetric AuxPDAs is equivalent to ours [All]. Hence, the following theorem is immediate.

Theorem 4.3. *(Allender and Lange [AL10]).* **SLogCFL = LogCFL**.

5 Realizability with one stack symbol

The realizability problems 1STREAL, 1USTREAL, 1DSTREAL and 1SGUSTREAL are obtained by restricting the previously defined realizability problems to use only one stack symbol i.e., by insisting that $k = 1$ in the above definitions. Since the vertices are all labeled with one label, we may omit the vertex labels in the definitions. After omitting the vertex labels, the corresponding *realizability* can be defined using a context-free language as shown below.

5.1 1STREAL, 1USTREAL and 1DSTREAL

1STREAL is the following graph realizability problem. We are given a directed graph $\mathcal{G}(V, E)$, with edges labeled from $\{push, pop, \epsilon\}$. The ordered pair (s, t), where $s, t \in V$, is said to be realizable if the following two conditions hold :

- There is a directed path (say P) from s to t.
- The concatenation of the edge labels on the path P is a string produced by the following context-free grammar : $S \rightarrow S\ S$; $S \rightarrow push\ S\ pop$; $S \rightarrow \epsilon$; $S \rightarrow \emptyset$. Here $\emptyset$ denotes the empty string.

1USTREAL and 1DSTREAL are defined analogously with the symmetric and deterministic restrictions.

5.2 1SGUSTREAL

1SGUSTREAL is the following graph realizability problem. We are given an undirected graph $\mathcal{G}(V, E)$, with the edges labeled from $\{push, pop, \epsilon\}$. The edge labels are "symmetric" as defined in Section 2.2. The pair (s, t), where $s, t \in V$, is said to be realizable if the following two conditions hold :

- There is an undirected path (say P) from s to t.
- The concatenation of the edge labels on the path P is a string produced by the following context-free grammar : $S \to S\ S$; $S \to push\ S\ pop$; $S \to pop\ S\ push$; $S \to \epsilon$; $S \to \emptyset$. Here $\emptyset$ denotes the empty string.

5.3 Polynomial length paths

1DSTREAL(POLY), 1SGUSTREAL(POLY), 1USTREAL(POLY) and 1STREAL(POLY) are defined analogously. Let **1LogDCFL**, **1SGSLogCFL**, **1SLogCFL** and **1LogCFL** be the corresponding complexity classes.

Theorem 5.1. 1DSTREAL and 1DSTREAL(POLY) are equivalent. **L** = **1LogDCFL**.

Proof. Recall the definition of DSTREAL (see Section 2.3). In the corresponding definition of 1DSTREAL there are no stack symbols. Hence the underlying graph has outdegree at most one for each vertex. In this graph any s-t path has at most n vertices. Hence 1DSTREAL and 1DSTREAL(POLY) are equivalent. Also, such a graph represents the configuration graph of a deterministic logspace Turing machine. Hence, deciding the existence of an s-t path in such graphs is **L**-complete. □

Theorem 5.2. **NL** = **1LogCFL**.

Proof. $\mathbf{1LogCFL} \subseteq \mathbf{NL}$: An **NL**-machine (say $\mathcal{M}$) non-deterministically guesses an s-t path (say P). $\mathcal{M}$ traverses the edges along P and maintains a counter C. $\mathcal{M}$ increments (resp. decrements) C if the current edge is labeled $push$ (resp. pop). If C was ever negative then $\mathcal{M}$ rejects. $\mathcal{M}$ accepts iff $C = 0$ when it reaches t.

$\mathbf{NL} \subseteq \mathbf{1LogCFL}$: We replace each directed edge (say (u, v)) of STCONN by two directed edges (u, w) and (w, v) and label them *push* and *pop* respectively. We add a new vertex w for each edge (u, v). There is an s-t path in the original graph iff there is a realizable path (according to the definition from Section 5.1) in the modified graph. □

6 Balanced Paths

In this section, We introduce two natural graph connectivity problems characterizing **1SGSLogCFL** and **1SLogCFL**.

Let $\mathcal{G}(V, E)$ be a directed graph. Let $\mathcal{G}'(V, E')$ be the underlying undirected graph of $\mathcal{G}$. Let P be a path in $\mathcal{G}'$. Let $e = (u, v)$ be an edge along the path P. Edge e is called *neutral* edge if both (u, v) and (v, u) are in E. Edge e is called *forward* edge if $(u, v) \in E$ and $(v, u) \notin E$. Edge e is called *backward* edge if $(u, v) \notin E$ and $(v, u) \in E$.

A path (say P) from $s \in V$ to $t \in V$ in $\mathcal{G}'(V, E')$ is called *balanced* if the number of forward edges along P is equal to the number of backward edges along P. A balanced path might have any number of neutral edges. By definition, if there is a balanced path from s to t then there is a balanced path from t to s. The path P may not be a simple path. We are concerned with balanced paths of length at most n.

BALANCED ST-CONNECTIVITY : Given a directed graph $\mathcal{G}(V, E)$ and two distinguished nodes s and t, decide if there is *balanced* path (of length at most n) between s and t.

Let P be a path from $s \in V$ to $t \in V$ in $\mathcal{G}(V, E)$. We say $v \in P$ if the vertex v is on the path P. For $v \in P$ we denote by P_v the subpath of P starting from s and ending at v. We say that P is *positive* if the number of forward edges of P_v is at least the number of backward edges of P_v, for all $v \in P$. In other words, the number of forward edges minus the number of backward edges of P_v is positive, for all $v \in P$. We say that P is *positive balanced* if P is positive and balanced. By definition, if there is a positive balanced path from s to t then there is a positive balanced path from t to s.

POSITIVE BALANCED ST-CONNECTIVITY : Given a directed graph $\mathcal{G}(V, E)$ and two distinguished nodes s and t, decide if there is *positive balanced* path (of length at most n) between s and t.

Theorem 6.1. BALANCED ST-CONNECTIVITY is **1SGSLogCFL**-complete.

Proof. BALANCED ST-CONNECTIVITY $\in$ **1SGSLogCFL**: Let $\mathcal{G}(V, E)$ be an instance of BALANCED ST-CONNECTIVITY. Let $\mathcal{G}'(V, E')$ be the underlying undirected graph of $\mathcal{G}$. If $(u, v) \in E$ and $(v, u) \in E$ then we label the edges (u, v) and (v, u) of $\mathcal{G}'$ with ϵ. If $(u, v) \in E$ and $(v, u) \notin E$ then we label the edge (u, v) of $\mathcal{G}'$ with $push$ and label the edge (v, u) of $\mathcal{G}'$ with pop. Note that the edge labels of $\mathcal{G}'$ are symmetric. There is a balanced s-t path in $\mathcal{G}$ iff there is a realizable s-t path (according to the definition from Section 5.2) in $\mathcal{G}'$.
BALANCED ST-CONNECTIVITY is **1SGSLogCFL**-hard: An instance of **1SGSLogCFL** is an undirected graph (say G) with edges labeled from $\{push, pop, \epsilon\}$. These edge labels are symmetric as defined in Section 2.2. We construct a directed graph H on the same vertex set. If the edge (u, v) of G is labeled ϵ we add the edges (u, v) and (v, u) in H. If the edge (u, v) is labeled $push$ (by symmetry the edge (v, u) is labeled pop) we add a directed edge u, v in H. There is a realizable s-t path in G iff there is a balanced s-t path in H.

□

Theorem 6.2. POSITIVE BALANCED ST-CONNECTIVITY is **1SLogCFL**-complete.

Proof. Similar to the proof of Theorem 6.1. □

Corollary 6.3. **L = 1LogDCFL $\subseteq$ 1SGSLogCFL $\subseteq$ 1SLogCFL $\subseteq$ 1LogCFL = NL**.

In a recent work [KS11], we proved the following theorem.

Theorem 6.4. *(Kintali, Shapira [KS11])* Let $G(V, E)$ be a directed graph with two distinguished vertices $s, t \in V$ and let P be a balanced path from s to t. Then there exists a balanced path P' from s to t such that the length of Q is $O(n^3)$.

Hence we have the following corollary.

Corollary 6.5. Both 1SGUSTREAL and 1SGUSTREAL(POLY) are **1SGSLogCFL**-complete.

Figure 1 summarizes the relationship among the above defined classes that lie between $\mathbf{L}$ and $\mathbf{LogCFL}$. A directed edge from class $\mathbf{A}$ to class $\mathbf{B}$ shows that $\mathbf{A} \subseteq \mathbf{B}$. In addition to the relations shown, $\mathbf{RL} \subseteq \mathbf{RLogCFL}$ and $\mathbf{BPL} \subseteq \mathbf{BPLogCFL}$. Recall that BALANCED ST-CONNECTIVITY is $\mathbf{1SGSLogCFL}$-complete and POSITIVE BALANCED ST-CONNECTIVITY is $\mathbf{1SLogCFL}$-complete.

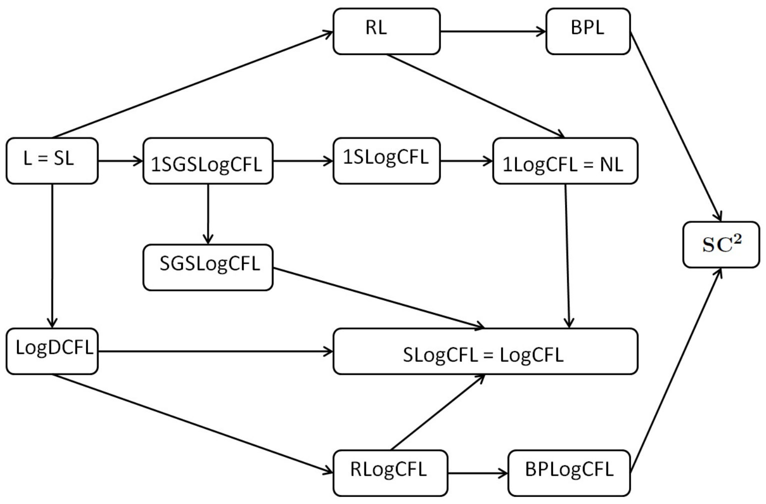

Figure 1: Relationship among the complexity classes lying between $\mathbf{L}$ and $\mathbf{LogCFL}$

7 Transitive Closure

The definitions and theorems in this section are applicable to all the graph realizability problems defined above. We present the definitions and theorems for STREAL, the most general graph realizability problem. We now define the transitive closure of an instance of STREAL.

Definition 7.1. Let $\mathcal{G} = \langle \Upsilon, \mathcal{E} \rangle$ be an instance of STREAL. The **transitive closure** of $\mathcal{G}$, denoted by $\mathcal{G}^* = \langle \Upsilon^*, \mathcal{E}^* \rangle$, is a pair of gap and standard matrix such that for all $a, b, c, d \in V$,
(i) $\mathcal{E}^*[a][b] = 1$ iff $(a \rightsquigarrow b)$ and
(ii) $\Upsilon^*[a, (c, d), b] = 1$ iff (a, b) is realizable with gap (c, d).

The transitive closure of STREAL(POLY) deals only with paths (realizable paths and paths with gap) of length at most n.

Definition 7.2. Let $\mathcal{G} = \langle \Upsilon, \mathcal{E} \rangle$ be an instance of STREAL(POLY). The **transitive closure** of $\mathcal{G}$, denoted by $\mathcal{G}^* = \langle \Upsilon^*, \mathcal{E}^* \rangle$, is a pair of gap and standard matrix such that for all $a, b, c, d \in V$,
(i) $\mathcal{E}^*[a][b] = 1$ iff $(a \rightsquigarrow b)$ and this realizable path from a to b is of length at most n and
(ii) $\Upsilon^*[a, (c, d), b] = 1$ iff $(a \rightsquigarrow (c, d) \rightsquigarrow b)$ and this *path with gap* is of length at most n. Recall the definition of the *length* of *path with gap* from Section 2.5

7.1 Tensor Products

We now present several tensor products acting on $\mathcal{E}$ and Υ. The products $\otimes_1$ to $\otimes_5$ are introduced in [Ven06] to study probabilistic AuxPDAs. We introduce $\otimes_6$ and $\otimes_7$. These products update the standard matrix $\mathcal{E}$

and the gap matrix Υ with new “connectivity information” of $\mathcal{G}$. Let $\mathcal{E}$, $\mathcal{E}_1$, $\mathcal{E}_2$ represent standard matrices and Υ, Υ_1, Υ_2 represent gap matrices. Let a, b, c, d, z represent the vertices of $\mathcal{G}$. Matrices indexed by two (resp. four) indices are standard (resp. gap) matrices. Since we are dealing with boolean matrices, all the summations (resp. multiplications) are interpreted as boolean $\vee$ (resp. boolean $\wedge$).

1. If $(a \leadsto z)$ and $(z \leadsto b)$ then $(a \leadsto b)$:
$$(\mathcal{E}_1 \otimes_1 \mathcal{E}_2)[a, b] = \sum_z \mathcal{E}_1[a, z] \cdot \mathcal{E}_2[z, b].$$
2. If $(a \leadsto (c, d) \leadsto b)$ and $(c \leadsto d)$ then $(a \leadsto b)$:
$$(\Upsilon \otimes_2 \mathcal{E})[a, b] = \sum_{c,d} \Upsilon[a, (c, d), b] \cdot \mathcal{E}[c, d].$$
3. If $(a \leadsto (c, d) \leadsto b)$ and $(b \leadsto z)$ then $(a \leadsto (c, d) \leadsto z)$:
$$(\Upsilon \otimes_3 \mathcal{E})[a, (c, d), z] = \sum_b \Upsilon[a, (c, d), b] \cdot \mathcal{E}[b, z].$$
4. If $(z \leadsto a)$ and $(a \leadsto (c, d) \leadsto b)$ then $(z \leadsto (c, d) \leadsto b)$:
$$(\mathcal{E} \otimes_4 \Upsilon)[z, (c, d), b] = \sum_a \mathcal{E}[z, a] \cdot \Upsilon[a, (c, d), b].$$
5. If $(a \leadsto (c, d) \leadsto b)$ and $(c \leadsto (e, f) \leadsto d)$ then $(a \leadsto (e, f) \leadsto b)$:
$$(\Upsilon_1 \otimes_5 \Upsilon_2)[a, (e, f), b] = \sum_{c,d} \Upsilon_1[a, (c, d), b] \cdot \Upsilon_2[c, (e, f), d].$$
6. If $(a \leadsto (c, d) \leadsto b)$ and $(z \leadsto d)$ then $(a \leadsto (c, z) \leadsto b)$:
$$(\Upsilon \otimes_6 \mathcal{E})[a, (c, z), b] = \sum_d \Upsilon[a, (c, d), b] \cdot \mathcal{E}[z, d].$$
7. If $(a \leadsto (c, d) \leadsto b)$ and $(c \leadsto z)$ then $(a \leadsto (z, d) \leadsto b)$:
$$(\Upsilon \otimes_7 \mathcal{E})[a, (z, d), b] = \sum_c \Upsilon[a, (c, d), b] \cdot \mathcal{E}[c, z].$$

All the above mentioned products are *sound*. For example, let $\mathcal{G} = \langle \Upsilon, \mathcal{E} \rangle$ and $\mathcal{G}' = \langle \Upsilon \otimes_3 \mathcal{E}, \mathcal{E} \rangle$. It is easy to see that there is an s-t realizable path in $\mathcal{G}$ if and only if there is an s-t realizable path in $\mathcal{G}'$. In the following subsection, we show that these products are *sufficient* to define a natural graph squaring operation.

7.2 Squaring Operation

Given $\mathcal{G} = \langle \Upsilon, \mathcal{E} \rangle$ the following algorithm computes the "square" of $\mathcal{G}$. Theorem 7.3 implies a natural polynomial time algorithm to solve STREAL(POLY). It plays a crucial role in the proofs of correctness of parallel and space efficient algorithms for $\mathbf{SGSLogCFL}$ (see Section 8 and Section 9).

```
$\mathbf{Square}(\langle \Upsilon, \mathcal{E} \rangle)$
    $\mathcal{E} = \mathcal{E} \otimes_1 \mathcal{E}$
    $\mathcal{E} = \Upsilon \otimes_2 \mathcal{E}$
    $\Upsilon = \Upsilon \otimes_3 \mathcal{E}$
    $\Upsilon = \mathcal{E} \otimes_4 \Upsilon$
    $\Upsilon = \Upsilon \otimes_5 \Upsilon$
    $\Upsilon = \Upsilon \otimes_6 \mathcal{E}$
    $\Upsilon = \Upsilon \otimes_7 \mathcal{E}$
return $\langle \Upsilon, \mathcal{E} \rangle$
```

Theorem 7.3. Let $\mathcal{G}$ be an instance of STREAL(POLY). $\mathcal{G}^* = \langle \Upsilon^*, \mathcal{E}^* \rangle$ can be computed using $O(\log n)$ repeated applications of $\mathbf{Square}(\mathcal{G})$.

Proof. We may assume that there are no ϵ edges in an instance of STREAL. This can be achieved by replacing each directed edge (u, v) labeled with ϵ with two directed edges (u, w) and (w, v), where w is a new node. The label of (u, w) (resp. (w, v)) is set to *push* (resp. *pop*). Repeat this for every edge, adding a new node every time. We introduce a new label α_{k+1} and label all the new nodes with α_{k+1}. It is easy to see that a path from s to t is realizable in the original graph if and only if it is realizable in the new graph. Hence STREAL reduces to STREAL with no ϵ edges.

Equivalently, we may assume that an AuxPDA always pushes or pops a symbol at every step. If an AuxPDA doesn't push or a pop at every step then we introduce an extra symbol in its stack alphabet which is pushed onto the stack when nothing is done to the stack. This alphabet is first popped before performing a valid stack move. In the rest of this proof we assume that all realizable paths are of even length.

We first state the relevant definitions and lemmas from [NR95]. A *path description* is a triple (A, B, i) consisting of two surface configurations A and B and an even natural number i. A description is *realizable* if A and B are realizable. In particular, (A, B, i) represents several paths of length i between A and B.

The relation $\vdash$ shows how to split computation paths recursively into shorter and shorter paths until we end up with trivial paths. Let $x = (A, B, i)$, $y = (C, D, j)$, and $z = (E, B, k)$ be path descriptions. Then we write $y, z \vdash x$ and $z, y \vdash x$ if and only if

(1) the level of the pushdown is equal for A, E and B;

(2) there exists a computation from A to C in one step, pushing a symbol α onto the pushdown tape during this step;

(3) there exists a computation from D to E in one step, popping α from the pushdown tape; and

(4) $j + k = i - 2$.

Note that identical pushdown heights of A, E and B imply that C and D have same pushdown height. Also, j and k are always even. In this way we can reduce the checking of realizability of x to the checking of the realizability of smaller paths y and z. We now state two crucial lemmas from [NR95] that gives a "balanced" partition of realizable computation. The proofs of these lemmas are based on a *recursive descent* using the properties of the decomposition relation $\vdash$.

Lemma 7.4. (Niedermeier and Rossmanith [NR95]) Let (A, B, i) denote a realizable path description for a fixed computation path of length $i \geq 2$ between A and B. Then there exist uniquely determined subpaths (C, D, i_1), (E, F, i_2) and (G, D, i_3) of (A, B, i) such that $(E, F, i_2), (G, D, i_3) \vdash (C, D, i_1)$ and $i_2, i_3 \leq i/2 < i_1$.

Lemma 7.4 splits a fixed computation path into three paths. The first two paths are the subpaths (E, F, i_2) and (G, D, i_3) and the third one is the path (A, B, i) *with gap* (C, D, i_1). This means that the verification of the realizability of (A, B, i) can be reduced to showing that (E, F, i_2), (G, D, i_3) and the pair-with-gap $(A, (C, D, i_1), B, i)$ are realizable.

A description for a *path with gap* $(A, (C, D, j), B, i)$ consists of four surface configurations A, B, C, D and two even numbers i and j with $j \leq i$. A path with gap $(A, (C, D, j), B, i)$ is called *realizable* iff $(A \rightsquigarrow (C, D) \rightsquigarrow B)$ and there exists a computation path from A to C and one from D to B with total number of steps $j - i$. Now we generalize the decomposition relation $\vdash$ to computation paths with gap. Let $x = (A, (C, D, j), B, i)$ and, first, let $y = (E, (C, D, j), F, k)$ and $z = (G, B, l)$ or, second, let $y = (E, F, k)$, $z = (G, (C, D, j), B, l)$. Then we write $y, z \vdash x$ and $z, y \vdash x$ if and only if

(1) the level of the pushdown is equal for A, G and B;

(2) there exists one step from A to E pushing a symbol α onto the pushdown tape;

(3) there is one step from F to G popping α from the pushdown tape; and

(4) $k + l = i - 2$.

The following lemma is the analogue of Lemma 7.4 for a fixed computation path with gap.

Lemma 7.5. (Niedermeier and Rossmanith [NR95]) Let $(A, (C, D, j), B, i)$, $i - j \geq 2$ denote a realizable path with gap. Then there exist uniquely determined paths $y = (E, (C, D, j), F, i_1)$ and either

(1) $z_1 = (G, (C, D, j), H, i_2)$ and $z_2 = (I, F, i_3)$, such that $z_1, z_2 \vdash y$ and $i_2 - j \leq (i - j)/2 < i_1 - j$ or

(2) $z_1 = (G, H, i_2)$ and $z_2 = (I, (C, D, j), F, i_3)$, such that $z_1, z_2 \vdash y$ and $i_3 - j \leq (i - j)/2 < i_1 - j$.

Lemma 7.5 is used to decompose *paths with gaps* in a balanced way. To check the realizability of $(A, (C, D, j), B, i)$ we examine the realizability of $(A, (E, F, i_1), B, i)$, z_1 and z_2. Both possible subpaths with gap have length less than or equal to half of the lenght of the whole path with gap $(A, (C, D, j), B, i)$. The arising subpath without gap may have a maximum length of $i - j - 2$ and will be split in a balanced way using Lemma 7.4.

We are now ready to prove our theorem. For realizable paths (both standard and gap paths) of length at most four, it is easy to verify that an application of **SimpleSquare** reduces the path length by a factor of at least $\frac{3}{4}$. For paths of length greater than four, we divide the path into three smaller paths using Lemma 7.4 for standard paths and Lemma 7.5 for path with gaps and use induction. This implies that one applcation of **SimpleSquare** reduces the path length by a constant factor. Hence $O(\log n)$ repeated applications of $\mathbf{SimpleSquare}(\mathcal{G})$ suffice to compute the transitive closure $\mathcal{G}^*$. $\square$

8 Parallel algorithms for SGSLogCFL

Let $\mathcal{G} = \langle \Upsilon, \mathcal{E} \rangle$ be an instance of **SGSLogCFL**. Let the vertices of $\mathcal{G}$ be $V = \{1, 2, \ldots, n\}$. $\mathcal{G}$ is represented by an $n \times n$ standard matrix $\mathcal{E}$ and an $n^2 \times n^2$ gap matrix Υ. In this section, we present parallel algorithms to compute $\mathcal{G}$'s transitive closure $\mathcal{G}^* = \langle \Upsilon^*, \mathcal{E}^* \rangle$. Let $V^2 = V \times V$ be the set of pairs of vertices. In the rest of this paper the term "vertex" refers to elements from V as well as V^2. Let $V^4 = V \times V \times V \times V$.

$\mathcal{G}$ has two types of edges. The standard edges from V^2 are present in $\mathcal{E}$ and the gap edges from V^4 are present in Υ. In the rest of this paper the term "edge" refers to elements from V^2 as well as V^4.

Definition 8.1. A subset of vertices $S \subseteq V$ is a **standard component** (s-component) of $\mathcal{G}$ iff for all $u, v \in S$ it holds that $(u \rightsquigarrow v)$ and $(v \rightsquigarrow u)$.

Definition 8.2. A subset $S \subseteq V^2$ is a **gap component** (g-component) of $\mathcal{G}$ iff for all $(a, b), (c, d) \in S$ it holds that $(a \rightsquigarrow (c, d) \rightsquigarrow b)$ and $(c \rightsquigarrow (a, b) \rightsquigarrow d)$.

In the rest of this paper the term "component" refers to both standard and gap components. If there is ambiguity we will explicitly say s-component or g-component.

A *pseudotree* $P = (C, D)$ is a maximal connected directed graph with $|C|$ vertices and $|D|$ arcs such that $|C| = |D|$ and each vertex has outdegree one. Note that every pseudotree has exactly one simple directed cycle (which may be a self-loop). The number of arcs in the directed cycle of a pseudoree P is called its *circumference*. A pseudotree whose cycle is a self-loop on some vertex r (called the *root*) is called a *rooted tree*. A *rooted star* R with root r, is a rooted tree whose arcs are of the form (x, r) with $x \in R$. A *pseudoforest* is a collection of pseudotrees.

Symmetric Squaring : We first present a simplified squaring algorithm when the input graph is an instance of $\mathbf{SGSLogCFL}$. Here the matrices $\mathcal{E}$ and Υ are symmetric i.e., $\mathcal{E}[a, b] = \mathcal{E}[b, a]$ and $\Upsilon[(a, b), (c, d)] = \Upsilon[(c, d), (a, b)]$. Moreover, $\Upsilon[(a, b), (c, d)] = \Upsilon[(a, b), (d, c)] = \Upsilon[(b, a), (c, d)] = \Upsilon[(b, a), (d, c)]$. Due to this symmetry, the products $\otimes_3$, $\otimes_4$, $\otimes_6$ and $\otimes_7$ are equivalent. Corollary 8.3 follows from Theorem 7.3.

```
**SymmetricSquare**($\langle \Upsilon, \mathcal{E} \rangle$)
    $\mathcal{E} = \mathcal{E} \otimes_1 \mathcal{E}$
    $\mathcal{E} = \Upsilon \otimes_2 \mathcal{E}$
    $\Upsilon = \Upsilon \otimes_3 \mathcal{E}$
    $\Upsilon = \Upsilon \otimes_5 \Upsilon$
**return** $\langle \Upsilon, \mathcal{E} \rangle$
```

Corollary 8.3. Let $\mathcal{G}$ be an instance of $\mathbf{SGSLogCFL}$. $\mathcal{G}^*$ can be computed using $O(\log n)$ repeated applications of $\mathbf{SymmetricSquare}(\mathcal{G})$.

8.1 An $O(\log^2 n)$ time parallel algorithm

We will assume that there is one processor P_i assigned to each vertex $i \in V$, one processor P_{ij} assigned to each edge $(i, j) \in V^2$ and one processor P_{ijkl} assigned to each *gap edge* $(i, j, k, l) \in V^4$. We use a vector $X_{\mathcal{E}}$ of length n to specify the s-components of $\mathcal{G}$ as follows : if $V_c \subseteq V$ is any s-component, then for all $i \in V_c$, $X_{\mathcal{E}}(i)$ equals the least element of V_c. We use an $n \times n$ matrix X_{Υ} to specify the g-components of $\mathcal{G}$ as follows : if $W_c \subseteq V^2$ is any g-component, then for all $(i, j) \in W_c$, $X_{\Upsilon}(i, j)$ equals the lexicographically least element of W_c.

The algorithm **Connect** iteratively computes the vectors $X_{\mathcal{E}}$ and X_{Υ} from the input $\mathcal{G} = \langle \Upsilon, \mathcal{E} \rangle$ and updates Υ^* and $\mathcal{E}^*$. It is based on a hook and contract algorithm [HCS79] that works as follows : Initially each element from V is an s-component by itself. Their edge-lists correspond to the undirected edges of $\mathcal{E}$. These components will eventually grow and become the corresponding s-components. Initially each element from V^2 is a g-component by itself. Their edge-lists correspond to the undirected edges of Υ. These components will eventually grow and become the corresponding g-components. The "components" at each stage of the algorithm are sets of "vertices" *found so far* to belong to the same (standard or gap) component of $\mathcal{G}$. Each component is equipped with a linked list of edges that connect it to other components. The algorithm repeats the following steps until there are no edges left :

1. Each component picks an edge pointing to a lexicographically minimum vertex from its edge-list leading to a neighboring component and hooks to it. If a component has an empty edge-list, it hooks to itself. The details of hooking are presented in **StandardHook** and **GapHook**. Note that both these hooking steps use the previously computed connectivity information from *both* Υ^* and $\mathcal{E}^*$. These hooking processes create clusters of components called pseudotrees. The s-components form pseudotrees on the vertex set V and g-components form pseudotrees on the vertex set V^2.

2. Each pseudotree is merged into a new component with one of its vertices as its representative. Each representative receives into its *new* edge-list all the edges contained in the edge-lists of its pseudotree. At this stage the matrices $\mathcal{E}^*$ and Υ^* are updated with "new" edges i.e., new connectivity information gathered from the hooking and contracting steps. Edges that are internal to the components are removed.

In the algorithm **Connect**, during the first iteration the edges connecting each vertex to neighboring vertices are examined (steps 6-11), and sets of vertices which are known to be connected are identified (steps 14-17). Each such set of vertices is merged into a "supervertex". These supervertices are specified by the vectors $X_{\mathcal{E}}(i)$ and $X_{\Upsilon}(i,j)$. For each i in a supervertex, $X_{\mathcal{E}}(i)$ equals the smallest-numbered vertex in the supervertex. For each (i,j) in a supervertex, $X_{\Upsilon}(i,j)$ equals the lexicographically first vertex in the supervertex. In succeeding iterations, the edges connecting each supervertex to neighboring supervertices are examined in steps 6-11, and sets of supervertices are merged in steps 14-17. The process continues until all the vertices in a (standard and gap) component have been merged into one gigantic supervertex.

Theorem 8.4. The algorithm **Connect** finds $\mathcal{G}^* = \langle \Upsilon^*, \mathcal{E}^* \rangle$ in parallel time $O(\log^2 n)$ using n^4 processors in the CREW PRAM model.

Proof. The following observations state that the hooking process creates pseudotrees on vertices from V and V^2.

Observation : Let $V_s \subseteq V$ denote an s-component of $\mathcal{G}$ such that $|V_s| \geq 2$ and define the function $C : V_s \to V_s$ by $C(i) = \textbf{StandardHook}(i)$. The function C defines a directed graph $G_s(C) = (V_s, F)$ where $F = \{(i, C(i)) \mid i \in V_s\}$. Then $G_s(C)$ is a collection of pseudotrees with circumference one, and the smallest-numbered vertex in each pseudotree is in the cycle of the pseudotree.

Observation : Let $V_g \subseteq V^2$ denote a g-component of $\mathcal{G}$ such that $|V_g| \geq 2$ and define the function $C : V_g \to V_g$ by $C(i,j) = \textbf{GapHook}(i,j)$. The function C defines a directed graph $G_g(C) = (V_g, F)$ where $F = \{((i,j), C(i,j)) \mid (i,j) \in V_g\}$. Then $G_g(C)$ is a collection of pseudotrees with circumference one, and the lexicographically smallest vertex in each pseudotree is in the cycle of the pseudotree.

The hooking processes (**StandardHook** and **GapHook**) and the contraction step are implemented to mimic the functionality of **SymmetricSquare**. Hence the correctness of the contraction step and the overall algorithm follows from Corollary 8.3.

```
Connect($\mathcal{G} = \langle \Upsilon, \mathcal{E} \rangle$)
 1: $\mathcal{E}^* \leftarrow \mathcal{E}$
 2: $\Upsilon^* \leftarrow \Upsilon$
 3: for all $i$ do $X_{\mathcal{E}}(i) = i$
 4: for all $i$ do $X_{\Upsilon}(i,j) = (i,j)$
 5: for $O(\log n)$ iterations do
 6:     for all $i$ do $Temp_{\mathcal{E}}(i) \leftarrow \mathbf{StandardHook}(i)$
 7:     for all $i$ do $Temp_{\mathcal{E}}(i) \leftarrow \min_j\{Temp_{\mathcal{E}}(j) \mid X_{\mathcal{E}}(j) = i \text{ and } Temp_{\mathcal{E}}(j) \neq i\}$
 8:     if none then $Temp_{\mathcal{E}}(i) \leftarrow X_{\mathcal{E}}(i)$
 9:     for all $i$ do $Temp_{\Upsilon}(i,j) \leftarrow \mathbf{GapHook}(i,j)$
10:     for all $i$ do $Temp_{\Upsilon}(i,j) \leftarrow \min_{(k,l)}\{Temp_{\Upsilon}(k,l) \mid X_{\Upsilon}(k,l) = (i,j) \text{ and } Temp_{\Upsilon}(k,l) \neq (i,j)\}$
11:     if none then $Temp_{\Upsilon}(i,j) \leftarrow X_{\Upsilon}(i,j)$
12:     for all $i$ do $X_{\mathcal{E}}(i) \leftarrow Temp_{\mathcal{E}}(i)$
13:     for all $(i,j)$ do $X_{\Upsilon}(i,j) \leftarrow Temp_{\Upsilon}(i,j)$
14:     for $O(\log n)$ iterations do
15:         for all $i$ do $Temp_{\mathcal{E}}(i) \leftarrow Temp_{\mathcal{E}}(Temp_{\mathcal{E}}(i))$
16:         for all $(i,j)$ do $Temp_{\Upsilon}(i,j) \leftarrow Temp_{\Upsilon}(Temp_{\Upsilon}(i,j))$
17:     end for
18:     for all $i$ do $X_{\mathcal{E}}(i) \leftarrow \min\{Temp_{\mathcal{E}}(i), X_{\mathcal{E}}(Temp_{\mathcal{E}}(i))\}$
19:     for all $(i,j)$ do $X_{\Upsilon}(i,j) \leftarrow \min\{Temp_{\Upsilon}(i,j), X_{\Upsilon}(Temp_{\Upsilon}(i,j))\}$
20:     for all $i,j$ do if $X_{\mathcal{E}}(i) = X_{\mathcal{E}}(j)$ then $\mathcal{E}^*[i,j] \leftarrow 1$.
21:     for all $i,j,k,l$ do if $X_{\Upsilon}(i,j) = X_{\Upsilon}(k,l)$ then $\Upsilon^*[i,(k,l),j] \leftarrow 1$.
22: end for
23: return $\mathcal{G}^* = \langle \Upsilon^*, \mathcal{E}^* \rangle$
```

```
StandardHook($i$)
 1: $S_1 \leftarrow \{X_{\mathcal{E}}(j) \mid \mathcal{E}^*[i,j] = 1 \text{ and } X_{\mathcal{E}}(j) \neq X_{\mathcal{E}}(i)\}$
 2: $S_2 \leftarrow \{X_{\mathcal{E}}(j) \mid \Upsilon^*[i,(k,k),j] = 1 \text{ and } X_{\mathcal{E}}(j) \neq X_{\mathcal{E}}(i)\}$
 3: $S = S_1 \cup S_2$
 4: if $S = \emptyset$ then
 5:     return $X_{\mathcal{E}}(i)$
 6: else
 7:     return $\min(S)$
 8: end if
```

```
GapHook(i, j)
1: $S_1 \leftarrow \{X_\Upsilon(k,l) \mid \Upsilon^*[i,(k,l),j] = 1$ and $X_\Upsilon(k,l) \neq X_\mathcal{E}(i,j)\}$
2: $S_2 \leftarrow \{X_\Upsilon(k,j) \mid \mathcal{E}^*[i,k] = 1$ and $X_\Upsilon(k,j) \neq X_\Upsilon(i,j)\}$
3: $S = S_1 \cup S_2$
4: if $S = \emptyset$ then
5:     return $X_\Upsilon(i,j)$
6: else
7:     return min($S$)
8: end if
```

Time and Processor Bounds : The main loop of the **Connect** program is executed $O(\log n)$ times. Within the loop, the iteration at step 14 is executed $O(\log n)$ times. Thus the algorithm requires $\Omega(\log^2 n)$ time. Steps 3, 12, 18 require $O(1)$ time using $\Omega(n)$ processors. Steps 4, 13, 19 require $O(1)$ time using $\Omega(n^2)$ processors. Steps 14-17 require $O(\log n)$ time using $\Omega(n^2)$ processors. **StandardHook** and **GapHook** are essentially computing minimum of at most $O(n^2)$ integers (accessing both $\mathcal{E}$ and Υ) and hence can be programmed to execute in $O(\log n)$ time using $O(n^2)$ processors. Hence the total running time is $O(\log^2 n)$. The total number of processors used is $O(n^4)$. □

Connect algorithm is a generalization of the parallel algorithm presented in [HCS79]. We added two hooking procedures (one for growing s-components and one for growing g-components). Unlike [HCS79] the new edges found after the contraction step are added in the matrices Υ^* and $\mathcal{E}^*$ *before* starting the next hooking step.

The algorithms of [JM97] and [CL95] can similarly be generalized to compute $\mathcal{G}^* = \langle \Upsilon^*, \mathcal{E}^* \rangle$ in parallel time $O(\log^{3/2} n)$ and $O(\log n \log\log n)$ respectively. The processor bounds in all these algorithms is polynomial in n. We now present an outline of the parallel algorithms of [JM97] and [CL95] and the necessary modifications to apply them to **SGSLogCFL**. We refer the reader to [JM97] and [CL95] for low-level implementation details of these algorithms.

8.2 An $O(\log^{3/2} n)$ time parallel algorithm

In the algorithm presented in the previous section, the size of the components formed after the hooking phase may vary a lot. A slow growing component may consist of as few as two vertices, whereas a fast growing component may have as many as n vertices in an s-component and n^2 vertices in a g-component. In order to allow the biggest component to contract to a single vertex, the contraction (steps 14-17) requires $\Theta(\log n)$ time. The slow-growing component may only double its size in each iteration. Hence, the algorithm must iterate $\log n$ times so that slow-growing components can eventually grow to its full size. A crucial observation made by Johnson and Metaxas [JM97] is that the slow-growing components need little time to contract and fast-growing components require fewer iterations to grow to their full size.

Johnson and Metaxas [JM97] presented an algorithm in which components are "scheduled" to hook and contract according to their *growth rate*. Their algorithm schedules every component to grow by a factor of at least $2^{\sqrt{\log n}}$ in a *phase* of $O(\log n)$ time. Hence, $\sqrt{\log n}$ phases suffice to find all the connected components in the graph. This implies an overall running time of $O(\log^{3/2} n)$. Within each phase the slow growing components are scheduled to hook and contract in $o(\log n)$ time repeatedly until they catch up with the fast growing components, whereas the fast growing components are left idle once they have achieved the intended size. We refer the reader to [JM97] for low-level implementation details of their algorithm. We now explain the main contributions of Johnson and Metaxas [JM97] and discuss how to apply their techniques to $\mathbf{SGSLogCFL}$. Since the matrices Υ and $\mathcal{E}$ are both symmetric matrices, the following techniques are applicable to $\mathbf{SGSLogCFL}$ with only minor changes to the **Connect** algorithm.

- In the algorithm of [HCS79] the vertices hook to a lexicographically minimum vertex. Hence the cycles in the pseudotrees are always self-loops. In Johnson-Metaxas algorithm vertices hook to the *first edge* in their edge-list. This creates pseudotrees of arbitrary circumference. These pseudotrees are to be contracted properly in the contraction phase. Johnson and Metaxas [JM97] introduced *cycle-reducing shortcutting* technique to solve this problem. This technique (i) contracts a pseudotree into a rooted tree in time logarithmic in its circumference, (ii) contracts a rooted tree into a rooted star in time logarithmic in the length of its longest path. We modify the **StandardHook** and **GapHook** to hook to the first edge in the edge list of the s-components and g-components respectively and apply the cycle-reducing shortcutting technique to the pseudotrees created by **StandardHook** and **GapHook** independently.

- There are potentially a large number of components that hook together in the hooking step. All these components are ready to give their edge-lists simultaneously to the new supercomponent's edge-list. It is expensive to compute the set of edges of all these components especially when concurrent writing is not allowed. Johnson and Metaxas [JM97] introduced *edge-plugging scheme* which achieves this objective in constant time. This technique uses the adjacency list representation of the input graph and maintains two *twin* copies of each edge to efficiently put all the edges previously belonging to the edge-lists of each vertex (of a pseudotree) into the edge-list of the newly created supervertex. We apply this edge-plugging scheme to the pseudotrees created by **StandardHook** and **GapHook** independently.

- There may be a large number of edges *internal* to a component. These internal edges cannot be used to find a mate (to hook) in the subsequent steps. Removing all the internal edges before picking an edge is expensive. Johnson and Metaxas [JM97] introduced a *growth-control schedule*. Components grow in size in a uniform way that controls their minimum sizes as long as continued growth is possible. The internal edges are identified and removed periodically to make the subsequent hooking steps more efficient. This growth control schedule is applied to both the s-components and g-components independently. We modify the **Connect** algorithm to run in $\sqrt{\log n}$ phases each running in $O(\log n)$ time. After each contraction step the newly found standard and gap edges are added in the matrices $\mathcal{E}^*$ and Υ^* respectively.

8.3 An $O(\log n \log\log n)$ time parallel algorithm

The Chong Lam algorithm [CL95] is also based on a hook and contract approach. Unlike the previous algorithm their algorithm creates pseudotrees without cycles i.e., rooted pseudotrees. The hooking schemes of [HCS79, JM97] may create pseudotrees with few vertices but a large degree. The hooking scheme of [CL95] guarantees that any pseudotree with a large degree must also contain a large number of vertices. This is achieved as follows : There is an ordering $<_d$ of the vertices such that $u <_d v$ iff the degree of u is less than the degree of v (or) the degrees are the same and u is less than v in their lexicographic ordering. Before every phase of the algorithm, every vertex is either active, inactive or done. All active and inactive vertices have nonzero degree, the done vertices have zero degree. The algorithm ensures that there are no multiedges between active vertices. The inactive vertices are always organized as a set of pseudotrees. In the beginning of the algorithm all vertices with nonzero degree are active, and the rest are done. The active vertices perform the following steps (in parallel). (i) if a vertex v has a neighbor larger (according to $<_d$) than itself, then v hooks to the largest such neighbor. (ii) if after the first step all neighbors of v are hooked to it, then v hooks to itself. Otherwise, if after the first step a neighbor u of v is hooked to a vertex different from v, then v hooks to u.

The representative vertex (of a pseudotree) is the *only* vertex in the pseudotree which is hooked to itself. In a contraction phase *some* of the pseudotrees are contracted to a representative vertex. A parameter is used to determine if a pseudotree is to be contracted. For every contracted pseudotree, its representative becomes a new active vertex and the rest of its vertices become done. The *done* vertices do not play any role for the rest of the algorithm. All multiedges between new active vertices are removed. The vertices of every uncontracted pseudotree become inactive.

The processing required by a hooking phase is performed (using pointer jumping) in parallel time $O(\log d)$, where d is the degree of the active vertex. Checking the degree of a hooking tree during the contraction phase is done (using pointer jumping and a constant time edge-list plugging technique) in parallel time $O(\log c)$, where c is the contraction parameter. The algorithm of Chong Lam (see **Connect** below) is a recursive algorithm. The size of the pseudotrees to be contracted varies with each recursive call. **MaxHook** and **Contract** are the hooking and contracting procedures explained above. Chong and Lam proved that a call to **Connect**($\lceil \log\log n \rceil$) contracts every connected component of the graph to a single vertex and all the other vertices are organized in a set of pseudotrees such that the root of the tree of a vertex u is the vertex to which the connected component of u is contracted.

```
**Connect**($k$)
    **MaxHook**;
    **if** $k > 0$ **then**
        **Connect**($2^{2^k}$)
        **Connect**($k-1$)
        **Connect**($k-1$)
    **Contract**($2^{2^{k+1}}$)
```

The modifications needed to generalize Chong Lam algorithm are very similar to those explained in Section 8.2. The hooking and contraction procedures are applied to the s-components and g-components separately. The new edges found after every contract operation are added to the matrices Υ^* and $\mathcal{E}^*$ and the new degrees of the vertices are recomputed.

As mentioned earlier the hooking procedure simulates the squaring procedure **SymmetricSquare**. To create the pseudotrees corresponding to s-components the hooking procedure accesses the gap matrix Υ also. Similarly to create the pseudotrees corresponding to g-components the hooking procedure accesses the standard matrix $\mathcal{E}$ also. This violates the exclusive read property of Chong Lam algorithm. We maintain two copies of $\mathcal{E}$ and Υ to implement the exclusive read property. We thus obtain an $O(\log n \log\log n)$ time EREW parallel algorithm computing $\mathcal{G}^* = \langle \Upsilon^*, \mathcal{E}^* \rangle$.

9 SGSLogCFL $\subseteq DSPACE(\log n \log\log n)$

Trifonov's algorithm [Tri08] is based on the above mentioned Chong Lam algorithm. A sequential algorithm is first derived from Chong Lam algorithm with minor changes to the hooking process. Each "step" of this sequential algorithm naturally defines a *configuration* of the algorithm, corresponding to a snapshot of the algorithm. All the edges incident to a given vertex are ordered and the hooking process is simulated sequentially. The recursive algorithm **Connect** (mentioned in Section 8.3) is converted into an $O(\log^2 n)$ space algorithm, which instead of storing all of its current configuration recomputes it whenever required. To simulate the edge-list plugging technique and pointer jumping techniques Trifonov made use of exploration walks on trees defined by Koucky [Kou02]. These walks allow efficient traversal of the pseudotrees. Since each vertex takes $\Theta(\log n)$ space and the vertices are stored locally at every level, this straighforward implementation takes an $O(\log^2 n)$ space.

To overcome this bottleneck all the functions are modified such that they never keep a vertex in their local variables. Instead of using vertices as arguments to functions, one current vertex is maintained in a global variable. All functions are implemented to return the required "information" about this vertex. The current vertex is an implicit argument to all functions. Only $\Theta(\log\log n)$ space is used to store the index of a bit of this current vertex. The concept of *current vertex* eliminates the need to store a vertex in a local variable. The introduction of one global current vertex and always returning information about this vertex is the main crucial idea behind Trifonov's algorithm. For more details of the algorithm and its pseudo-code see [Tri08].

Trifonov's algorithm finds the transitive closure of a given symmetric 0-1 matrix. An instance of $\mathbf{SGSLogCFL}$ consists of two symmetric 0-1 matrices, $\mathcal{E}$ and Υ. We implement two sequential algorithms similar to Trifonov's algorithm, one each for $\mathcal{E}$ and Υ. The algorithm running on the standard matrix is the *main* algorithm. It simulates the hooking and contracting steps of Chong Lam algorthm outlined in the previous section. The hooking processs needs access to the gap matrix Υ also. Whenever a "new" gap edge is required, the required entries of the gap matrix Υ are recomputed using the hooking and contracting functions acting on Υ. Thus the exploration walks and the manipulation of the *current standard* and *current gap* vertices are performed on both the s-components and g-components. At the end of all recursive calls the *main* function determines whether there is a *realizable path with symmetric gap* (recall the definition from Section 3) between two given vertices s and t.

Theorem 9.1. Let $\mathcal{G} = \langle \Upsilon, \mathcal{E} \rangle$ be an instance of SGUSTREAL(POLY). $\mathcal{G}^* = \langle \Upsilon^*, \mathcal{E}^* \rangle$ can be computed deterministically in $O(\log n \log\log n)$ space i.e., $\mathbf{SGSLogCFL} \subseteq DSPACE(\log n \log\log n)$.

Corollary 9.2. BALANCED ST-CONNECTIVITY $\in DSPACE(\log n \log\log n)$.

10 New Research Directions

Several interesting new research directions arise from our work :

1. BALANCED ST-CONNECTIVITY and POSITIVE BALANCED ST-CONNECTIVITY are natural graph connectivity problems lying between $\mathbf{L}$ and $\mathbf{NL}$. Studying their space complexity is an interesting research direction towards improving the space complexity of STCONN. In particular, it would be interesting to improve Theorem 9.1. Is $\mathbf{SGSLogCFL} \in \mathbf{L}$? Less ambitiously, is $\mathbf{SGSLogCFL} \in \mathbf{SC^2}$? See Section 1.2 for detailed discussion.

2. An alternate proof of Theorem 9.1 using the techniques of [RVW02, Rei08] or [RV05] seems to be a challenging task.

3. We studied SGUSTREAL purely as a graph connectivity problem. Is there a machine (or circuit) characterization of $\mathbf{SGSLogCFL}$? What is the relationship between (i) $\mathbf{SGSLogCFL}$ and $\mathbf{NL}$?

(ii) **SGSLogCFL** and **LogDCFL** ? (iii) **SGSLogCFL** and **DET**[2] ?

4. We know that DSTREAL = USTREAL = STREAL = **P**. We proved that SGUSTREAL(POLY) $\in DSPACE(\log n \log\log n)$. What is the complexity of SGUSTREAL ? Is it **P**-complete ?

5. In [KS11] we proved that 1SGUSTREAL = 1SGUSTREAL(POLY). Is 1USTREAL = 1USTREAL(POLY) ? Is 1STREAL = 1STREAL(POLY) ?

6. Allender and Lange [AL10] proved that **SLogCFL** = **LogCFL**. Is **1SLogCFL** = **1LogCFL** ? i.e., is POSITIVE BALANCED ST-CONNECTIVITY **NL**-complete ?

7. Cook proved that **LogDCFL** $\subseteq$ **SC^2** [Coo79]. This is the best known upper bound for **LogDCFL** for the last three decades. As noted earlier, DSTREAL(POLY) is **LogDCFL**-complete. Can our techniques be applied to DSTREAL(POLY) to simulate **LogDCFL** using $o(\log^2 n)$ space ?

8. **SLogCFL** vs **LogDCFL** : In the logspace setting we have **L** = **SL** $\subseteq$ **NL**. In the **LogCFL** setting, we have **LogDCFL** $\subseteq$ **SLogCFL** = **LogCFL** (see Theorem 4.3). By definition, we have **NL** $\subseteq$ **LogCFL**. It is known that **LogDCFL** $\subseteq$ **SC^2** [Coo79]. This motivates the study of the relationship between **LogDCFL** and **SLogCFL**. It would be interesting to understand what concepts are required to generalize the techniques of [RVW02, Rei08] to prove **LogDCFL** = **SLogCFL**. This would imply **NL** $\subseteq$ **SC^2**, i.e., STCONN can be solved by a deterministic algorithm in polynomial time and $O(\log^2 n)$ space.

9. **SLogCFL** vs **RLogCFL** : We have **LogDCFL** $\subseteq$ **SLogCFL** = **LogCFL** and **LogDCFL** $\subseteq$ **RLogCFL** $\subseteq$ **LogCFL** implying **RLogCFL** $\subseteq$ **SLogCFL**. In the logspace setting, prior to Reingold's work, Aleliunas et. al. [AKL+79] proved that **SL** $\subseteq$ **RL**, using random walks. It would be interesting to generalize their techniques to prove **SLogCFL** $\subseteq$ **RLogCFL**. Since **BPLogCFL** $\subseteq$ **SC^2** [Ven06], a proof of **SLogCFL** $\subseteq$ **RLogCFL** would imply **NL** $\subseteq$ **SC^2**. For more details see Section 1.2.

Acknowledgements : This project is partially funded by the NSF grant CCF-0902717. I gratefully acknowledge helpful discussions with Eric Allender, Klaus-Jörn Lange, Nutan Limaye, Richard J. Lipton, H. Venkateswaran, Dieter van Melkebeek and Anna Gal.

References

[AKL+79] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, Lászlo Lovász, and Charles Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. *FOCS*, pages 218–223, 1979. 3, 4, 24

[AL10] Eric Allender and Klaus-Jörn Lange. Symmetry coincides with nondeterminism for time-bounded auxiliary pushdown automata. *To appear in 25th Computational Complexity Conference*, 2010. 10, 24

[All] Eric Allender. Personal communication. 10

[2]**DET** is the class of problems **NC^1** Turing reducible to the determinant [Coo85].

[All07] Eric Allender. Reachability problems: An update. In *CiE*, pages 25–27, 2007. 1

[AP87] Foto N. Afrati and Christos H. Papadimitriou. The parallel complexity of simple chain queries. In *PODS*, pages 210–213, 1987. 2

[ATWZ00] Roy Armoni, Amnon Ta-Shma, Avi Wigderson, and Shiyu Zhou. An $o(\log^{4/3} n)$ space algorithm for (*s, t*) connectivity in undirected graphs. *J. ACM*, 47(2):294–311, 2000. 3

[BCD$^+$89] Allan Borodin, Stephen A. Cook, Patrick W. Dymond, Walter L. Ruzzo, and Martin Tompa. Two applications of inductive counting for complementation problems. *SIAM J. Comput.*, 18(3):559–578, 1989. 2, 3

[Bor77] Allan Borodin. On relating time and space to size and depth. *SIAM J. Comput.*, 6(4):733–744, 1977. 2

[CHL99] Ka Wong Chong, Yijie Han, and Tak Wah Lam. On the parallel time complexity of undirected connectivity and minimum spanning trees. In *SODA*, pages 225–234, 1999. 3

[CL95] Ka Wong Chong and Tak Wah Lam. Finding connected components in $o(\log n \log\log n)$ time on the erew pram. *J. Algorithms*, 18(3):378–402, 1995. 3, 20, 21

[Coo71] Stephen A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM*, 18(1):4–18, 1971. 1, 2, 5, 7, 28

[Coo79] Stephen A. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *STOC*, pages 338–345, 1979. 24

[Coo85] Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1-3):2–21, 1985. 24

[GHR95] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. Limits to parallel computation: P-completeness theory. *Oxford University Press*, 1995. 2

[Gre73] Sheila A. Greibach. The hardest context-free language. *SIAM J. Comput.*, 2(4):304–310, 1973. 2

[GW96] Anna Gál and Avi Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Struct. Algorithms*, 9(1-2):99–111, 1996. 2

[Har78] Michael A. Harrison. Introduction to formal languages theory. *Addison-Wesley series in computer science*, 1978. 2

[HCS79] Daniel S. Hirschberg, Ashok K. Chandra, and Dilip V. Sarwate. Computing connected components on parallel computers. *Commun. ACM*, 22(8):461–464, 1979. 3, 17, 20, 21

[Hen77] F. Hennie. Introduction to computability. *Addison-Wesley, Reading, MA*, 1977. 28

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17:935–938, 1988. 2

[JM95] Donald B. Johnson and Panagiotis Takis Metaxas. A parallel algorithm for computing minimum spanning trees. *J. Algorithms*, 19(3):383–401, 1995. 3

[JM97] Donald B. Johnson and Panagiotis Takis Metaxas. Connected components in $o(\log^{3/2} n)$ parallel time for the crew pram. *J. Comput. Syst. Sci.*, 54(2):227–242, 1997. 3, 20, 21

[Kou02] Michal Koucký. Universal traversal sequences with backtracking. *J. Comput. Syst. Sci.*, 65(4):717–726, 2002. 23

[KS11] Shiva Kintali and Asaf Shapira. On the Lengths of Realizable Paths. *Under Preparation*, 2011. 9, 12, 24

[KW93] Mauricio Karchmer and Avi Wigderson. On span programs. In *Structure in Complexity Theory Conference*, pages 102–111, 1993. 3

[Lim05] Nutan Limaye. Parallel complexity classes centered around LogCFL. *M.Sc. thesis, Anna University*, 2005. 1

[LP82] Harry R. Lewis and Christos H. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19:161–187, 1982. 2, 3, 29

[MR00] David Melski and Thomas W. Reps. Interconvertibility of a class of set constraints and context-free-language reachability. *Theoretical Computer Science*, 248(1-2):29–98, 2000. 2

[MRV99] Pierre McKenzie, Klaus Reinhardt, and V. Vinay. Circuits and context-free languages. In *COCOON*, pages 194–203, 1999. 2

[Nis94] Noam Nisan. RL $\subseteq$ SC. *Computational Complexity*, 4:1–11, 1994. 2, 4

[NR95] Rolf Niedermeier and Peter Rossmanith. Unambiguous auxiliary pushdown automata and semi-unbounded fan-in circuits. *Information and Computation*, 118(2):227–245, 1995. 2, 7, 15, 16

[NSW92] Noam Nisan, Endre Szemerédi, and Avi Wigderson. Undirected connectivity in O(log$^{3/2}$n) space. In *FOCS*, pages 24–29, 1992. 3

[NT95] Noam Nisan and Amnon Ta-Shma. Symmetric logspace is closed under complement. *STOC*, pages 140–146, 1995. 3

[Rei08] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008. 3, 4, 23, 24

[Rep96] Thomas W. Reps. On the sequential nature of interprocedural program-analysis problems. *Acta Inf.*, 33(8):739–757, 1996. 2

[Ruz80] Walter L. Ruzzo. Tree-size bounded alternation. *J. Comput. Syst. Sci.*, 21(2):218–235, 1980. 1, 2, 10

[RV05] Eyal Rozenman and Salil P. Vadhan. Derandomized squaring of graphs. In *APPROX-RANDOM*, pages 436–447, 2005. 3, 4, 23

[RVW00] Omer Reingold, Salil P. Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *FOCS*, pages 3–13, 2000. 3

[RVW02] Omer Reingold, Salil P. Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. *Annals of Mathematics*, 155(1), 2002. 3, 4, 23, 24

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. 1

[SBV10] Derrick Stolee, Chris Bourke, and N. V. Vinodchandran. A log-space algorithm for reachability in planar acyclic digraphs with few sources. In *IEEE Conference on Computational Complexity*, pages 131–138, 2010. 1

[Sud78] Ivan Hal Sudborough. On the tape complexity of deterministic context-free languages. *J. ACM*, 25(3):405–414, 1978. 1, 2, 9

[SV10] Derrick Stolee and N. V. Vinodchandran. Space-efficient algorithms for reachability in surface-embedded graphs. *Electronic Colloquium on Computational Complexity*, TR10-154, 2010. 1

[SZ99] Michael E. Saks and Shiyu Zhou. $BP_HSPACE(S) \subseteq DSPACE(S^{3/2})$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999. 3

[Sze87] Róbert Szelepcsényi. The method of forcing for nondeterministic automata. *Bulletin of EATCS*, 33:96–100, 1987. 2

[Tri08] Vladimir Trifonov. An O($\log n \log\log n$) space algorithm for undirected st-connectivity. *SIAM J. Comput.*, 38(2):449–483, 2008. 1, 3, 22, 23

[UG86] Jeffrey D. Ullman and Allen Van Gelder. Parallel complexity of logical query programs. In *FOCS*, pages 438–454, 1986. 2

[Ven91] H. Venkateswaran. Properties that characterize LogCFL. *J. Comput. Syst. Sci.*, 43(2):380–404, 1991. 1, 10

[Ven06] H. Venkateswaran. Derandomization of probabilistic auxiliary pushdown automata classes. *IEEE Conference on Computational Complexity*, pages 355–370, 2006. 2, 13, 24

[Ven09] H. Venkateswaran. Derandomization of probabilistic auxiliary pushdown automata classes. *Georgia Tech, College of Computing Technical Report GT-CS-09-06*, 2009. 2

[Wig94] Avi Wigderson. NL/poly $\subseteq \oplus$L/poly (preliminary version). In *Structure in Complexity Theory Conference*, pages 59–62, 1994. 2

Appendix

A Symmetric AuxPDA's

An **auxiliary pushdown automaton** (AuxPDA) is a multi-tape Turing machine with a two-way read-only input tape, a pushdown tape, and one or more work tapes. The pushdown alphabet has a distinguished symbol (say \$) which is initially *pushed* on the pushdown tape. The machine is designed so that the pushdown head never shifts left of \$ or changes \$. Further, the pushdown head can never shift left when scanning any tape symbol unless it first erases (i.e., pops) that symbol, and it can never shift right from a square unless it first prints (i.e., pushes) a nonblank symbol on that square. Space on an AuxPDA is the space used on the work tapes without counting the space on the pushdown tape. Formally, an AuxPDA is an 8-tuple

$M = (Q, \Sigma, \Sigma_0, \Sigma_\alpha, l, \Delta, s, F)$, where Q is a finite set of states, Σ is a finite tape alphabet, $\Sigma_0 \subseteq \Sigma$ is the input alphabet, $\Sigma_\alpha \subseteq \Sigma$ is the pushdown alphabet, l is the number of tapes, $s \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and Δ is a finite set of transitions.

We first define the *transition* of an AuxPDA that enable the AuxPDA to "peek" one square right or left on the input and work tapes and one square below the top symbol of the pushdown tape while changing its configuration. A transition is of the form $(p, \mathcal{S}, t_1, \ldots, t_l, q)$, where p and q are states, $\mathcal{S}$ is a stack triple, l is the number of tapes, and $t_1, \ldots, t_l$ are tape triples. A stack triple is either of the form (i) $(\alpha_a\alpha_b, P, \alpha_c\alpha_d)$, where $\alpha_a, \alpha_b, \alpha_c, \alpha_d \in \Sigma_\alpha$ and P is +1 or -1 ; or is of the form (ii) $(\alpha_a, 0, \alpha_b)$, where $\alpha_a, \alpha_b \in \Sigma_\alpha$. A tape triple is either of the form (i) (ab, D, cd), where $a, b, c, d \in \Sigma$ and D is +1 or -1; or is of the form (ii) $(a, 0, b)$, where $a, b \in \Sigma$.

A transition of the form $(p, \mathcal{S}, t_1, \ldots, t_l, q)$ signifies that M moves from state p to state q according to the stack and tape triples. The tape triple $t_i = (ab, +1, cd)$ signifies that when M is scanning symbol a on tape t_i, and with the square just to the right of the scanned square containing symbol b, M may rewrite these two squares to contain symbols c and d, respectively, move its tape head one square to the right. Similarly, a transition $(ab, -1, cd)$ signifies a potential left movement of the tape head, except that now the scanned symbol must be b and the one to its left a and these are rewritten as d and c, respectively. The tape triple $(a, 0, b)$ signifies that M replaces the symbol a with b without moving its head position. The stack triple is defined analogously with $P = +1$ (resp. $P = -1$) corresponding to a push (resp. pop) operation on the pushdown tape.

The **surface configuration** (introduced by Cook [Coo71]) of an AuxPDA on an input w consists of the state, contents and head positions of the work tapes, the head position of the input tape and the topmost symbol of the stack. Note that for a space $S(n)$-bounded AuxPDA, its surface configurations take only $O(S(n))$ space. In the rest of this section, we will refer to surface configurations as configurations. Let $\mathcal{C}(M)$ denote the set of all configurations of M. For an input w, and $C_1, C_2 \in \mathcal{C}(M)$ we write $C_1 \vdash_M C_2$ to denote that C_1 "yields" C_2. A *computation* by M is a sequence $C_0 \vdash_M C_1 \vdash_M \ldots \vdash_M C_n$, where $n \geq 0$ and $C_0, \ldots, C_n \in \mathcal{C}(M)$. The reflexive, transitive closure of $\vdash_M$ is denoted by $\vdash_M^*$ and the transitive closure is denoted by $\vdash_M^+$. An AuxPDA M is nondeterministic (resp. deterministic) if $\vdash_M$ is multi-valued (resp. single-valued).

Since the tape triples and stack triples of M enable it to peek into only a *constant* number of symbols, M can be simulated by a standard AuxPDA extending the notion of *big-headed* Turing machines [Hen77]. The "peeking" version of M enables us to define symmetric computation. Each transition $\delta = (p, \mathcal{S}, t_1, \ldots, t_l, q)$ has an inverse $\delta^{-1} = (q, \mathcal{S}^{-1}, t_1^{-1}, \ldots, t_l^{-1}, p)$ where if $\mathcal{S} = (\alpha, P, \beta)$ then $\mathcal{S}^{-1} = (\beta, -P, \alpha)$ and for $i = 1, \ldots, k$ if $t_i = (a, D, b)$ then $t_i^{-1} = (b, -D, a)$.

The inverse of an AuxPDA $M = (Q, \Sigma, \Sigma_0, \Sigma_\alpha, l, \Delta, s, F)$ is $M^{-1} = (Q, \Sigma, \Sigma_0, \Sigma_\alpha, l, \Delta^{-1}, s, F)$, where $\Delta^{-1} = \{\delta^{-1} : \delta \in \Delta\}$. An AuxPDA is *symmetric* if it is its own inverse i.e., if $\delta^{-1} \in \Delta$ whenever $\delta \in \Delta$. The symmetric closure of an AuxPDA $M = (Q, \Sigma, \Sigma_0, \Sigma_\alpha, l, \Delta, s, F)$ is $\overline{M} = (Q, \Sigma, \Sigma_0, \Sigma_\alpha, l, \Delta \cup \Delta^{-1}, s, F)$. Note that the symmetric closure of an AuxPDA is symmetric and a symmetric AuxPDA is its own symmetric closure. We now define the complexity class **SLogCFL**.

SLogCFL is the class of languages accepted by log space bounded and polynomial time bounded symmetric AuxPDA.

Let $\#$ be a new special symbol in the tape alphabet that does not belong to input alphabet. For an AuxPDA M, $M^\#$ is its normal form such that (1) M and $M^\#$ accept the same language in the same space bound, and have the same number of tapes; (2) $M^\#$ has no transitions into its initial state or out of any final state; (3) for any configurations $C_1, C_2 \in \mathcal{C}(M^{\#})$ if $C_1 \vdash_{M^{\#}} C_2$ then $|C_1| \leq |C_2|$, where $|C|$ represents the space of C. $M^{\#}$ is constructed from M by adding a new initial state and transitions from it to the old initial state; eliminating any transitions out of final states; and introducing a new *pseudoblank* symbol which $M^{\#}$ writes instead of writing (or rewriting) a blank on a worktape, and which $M^{\#}$ treats as indistinguishable from a blank when seen on a worktape. $\overline{M^{\#}}$ is the symmetric closure of $M^{\#}$.

The following lemma is proved by Lewis and Papadimitriou [LP82] in the context of symmetric Turing machines. By our definition of symmetric AuxPDA's, its proof follows by treating the "configurations" of a symmetric Turing machine as the "surface configurations" of a symmetric AuxPDA and augmenting the transitions with stack triples. We skip its proof since it is essentially the proof of [LP82].

Lemma A.1. Let $M = (Q, \Sigma, \Sigma_0, \Sigma_\alpha, l, \Delta, s, F)$ be any AuxPDA, and let $\mathcal{A} \subseteq \mathcal{C}(M)$. Suppose that

(a) for any $A_1, A_2 \in \mathcal{A}$, if $A_1 \vdash_M^{+\mathcal{A}} A_2$ then $A_2 \vdash_M^{+\mathcal{A}} A_1$

(b) for any $A \in \mathcal{A} \cup \mathcal{I}(M)$, and $B \notin \mathcal{A}$ and any C_1, C_2, C_3, if $A \vdash_M^{*\mathcal{A}} C_1 \dashv_M^{*\mathcal{A}} C_2 \dashv_M B \vdash_M C_3$, then $C_2 = C_3$

(c) for any $A_1 \in \mathcal{A} \cup \mathcal{I}(M)$, any $A_2 \in \mathcal{A}$, and any B, if $A_1 \vdash_M^{*\mathcal{A}} B \dashv_M^{*\mathcal{A}} A_2$ then $A_1 = A_2$.

Then $\overline{M^{\#}}$ accepts the same language as M in the same space as M.

Theorem A.2. $\mathbf{LogDCFL} \subseteq \mathbf{SLogCFL} \subseteq \mathbf{LogCFL}$.

Proof. Let M be a deterministic logspace bounded AuxPDA accepting a language $L \in \mathbf{LogDCFL}$. Then M satisfies the hypothesis of Lemma A.1, with $\mathcal{A} = \emptyset$. M satisfies the hypothesis (a) and (c) trivially. Since M is deterministic it satisfies the hypothesis (b). Hence $\overline{M^{\#}}$ accepts L. Hence, $\mathbf{LogDCFL} \subseteq \mathbf{SLogCFL}$. The second inclusion is trivial, since nondeterminism is more general than symmetry. □

It is routine to check that the AuxPDA thus constructed, satisfies the properties of Lemma A.1 and the following theorem is immediate.

Theorem A.3. USTREAL(POLY) is $\mathbf{SLogCFL}$-complete.