Electronic Colloquium on Computational Complexity, Report No. 21 (2002)

Space Efficient Algorithms for Directed Series-Parallel Graphs[1]

Andreas Jakoby[2] Maciej Liśkiewicz[3] Rüdiger Reischuk

Institut für Theoretische Informatik
Universität zu Lübeck
Wallstr. 40, D-23560 L ̈ubeck, Germany
`jakoby/liskiewi/reischuk@tcs.mu-luebeck.de`

March 2002

Abstract

The subclass of directed *series-parallel graphs* plays an important role in computer science. Whether a given graph is series-parallel is a well studied problem in algorithmic graph theory, for which fast sequential and parallel algorithms have been developed in a sequence of papers. Also methods are known to solve the reachability and the decomposition problem for series-parallel graphs time efficiently. However, no dedicated results have been obtained for the space complexity of these problems when restricted to series-parallel graphs – the topic of this paper.

Deterministic algorithms are presented for the *recognition, reachability, decomposition* and the *path counting problem* for series-parallel graphs that use only logarithmic space. Since for arbitrary directed graphs reachability and path counting are believed not to be solvable in Logspace the main contribution of this work are novel deterministic path finding routines that work correctly in series-parallel graphs, and a characterization of series-parallel graphs by forbidden subgraphs that can be tested space-efficiently. The space bounds are best possible, i.e. the decision problem is shown to be $\mathcal{L}$-complete with respect to $\mathcal{AC}^0$-reductions. They have also implications for the parallel time complexity of these problems when restricted to series-parallel graphs.

Finally, we sketch how these results can be generalised to extension of the series-parallel graph family: to graphs with multiple sources or multiple sinks and to the class of *minimal vertex series-parallel graphs.*

CLASSIFICATION: effi cient algorithms, algorithmic graph theory, computational complexity

[1]A preliminary version of these results has been presented at 18. STACSS'2001, Springer LNCS 2010, 339-352
[2]Part of this research was done while visiting the Depart. of Computer Science, Univ. of Toronto, Canada
[3]On leave from Instytut Informatyki, Uniwersytet Wrocławski, Poland.

ISSN 1433-8092

1 Introduction

In this paper, all graphs $G = (V, E)$ considered are directed. For two nodes u, v there may exist several edges connecting them, in this case we will differentiate them by $(u, v)_1$, $(u, v)_2, \ldots$. n denotes the number of vertices V of G and m the number of edges E. A well studied subclass of graphs are *series-parallel graphs,* for which different defi nitions and characterizations have been given [6]. We will consider the basic class, sometimes also called *two terminal series-parallel graphs,* that are most important for applications in program analysis.

Definition 1 *$G = (V, E)$ is a* **series-parallel graph, SP-graph** *for short, if either G is a line graph of length 1, that is a pair of nodes connected by a single edge, or there exist two series-parallel graphs $G_i = (V_i, E_i)$, $i = 1, 2$, with exactly one source $v_{in,i}$, and one sink $v_{out,i}$ such that $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, and either*

(A) **parallel composition***: $v_{in} = v_{in,1} = v_{in,2}$ and $v_{out} = v_{out,1} = v_{out,2}$, or*

(B) **series composition***: $v_{in} = v_{in,1}$ and $v_{out} = v_{out,2}$ and $v_{out,1} = v_{in,2}$.*

The node sets of the graphs G_i have to be disjoint except for the sources and sinks as required by the composition; the edge sets have to be completely disjoint.

Since in a parallel or series composition the sources and sinks of the two graphs G_i are merged the resulting graph G again has a unique source and a unique sink.

We assume that the graphs considered are specifi ed by a list of edges, but we put no restrictions on the ordering of the edges. In particular, it is not required that this ordering reflects the structure of the series-parallel composition operations. Otherwise, recognizing and handling series-parallel graphs becomes quite easy. The correctness and effi ciency of the algorithms presented below will not depend on the representation of the input graphs. For example, one could use adjacency-matrices as well.

Series-parallel graphs are suitable to describe the information flow within a program that is based on sequential and parallel composition. The graphical description of a program helps to decide whether it can be parallelised and to generate schedules for a parallel execution.

To determine whether a given graph G belongs to the class of series-parallel graphs is a basic problem in algorithmic graph theory. An optimal linear time sequential algorithm for this problem has been developed by Valdes, Tarjan, and Lawler in [17] long time ago. Also, fast parallel algorithms have been published. He and Yesha have presented an EREW PRAM algorithm working in time $O(\log^2 n)$ while using $n + m$ processors [13]. Eppstein has reduced the time bound constructing an algorithm that takes only $O(\log n)$ steps on the stronger PRAM model with concurrent instead of exclusive read and write, that requires $C(m, n)$ processors [11]. Here $C(m, n)$ denotes the number of processors necessary to compute the connected components of a graph in logarithmic time. Finally, Bodlaender and de Fluiter have presented an EREW PRAM algorithm using $O(\log n \cdot \log^* n)$ time and $O(n + m)$ operations [5].

The space complexity of this problem, however, has not been determined precisely so far. In this paper we give an answer to this question.

The *decomposition of a series-parallel graph* is useful to decide other graph properties. Hence, another important task is to compute such a decomposition effi ciently. In [17] a linear-time sequential algorithm for decomposing series-parallel graphs has been given. We will show that this task can be done in small space as well.

For general graphs, the *reachability problem,* that is the question whether there exists a path between a given pair of nodes, is the classical $\mathcal{NL}$-complete problem, also called GAP. For the parallel time complexity one can infer a logarithmic upper bound on CRCW PRAMs by well known simulations. The reachability problem restricted to series-parallel graphs, however, can be solved in logarithmic time already by an EREW PRAM using the minimal number $(n+m)/\log n$ of processors [16]. Certain graph properties like acyclicity are also complete for $\mathcal{NL}$, while for other problems their computational complexity is still unsolved. Recently, Allender and Mahajan have made a major step in classifying the computational complexity of planarity testing showing that this problem is hard for $\mathcal{L}$ and belongs to $\mathcal{SL}$ (symmetric Logspace) [3]. They leave as an open problem to close the gap between the lower bound and the upper bound. In this paper we determine the computational complexity of a nontrivial subproblem of planarity testing precisely. The question whether a graph is series-parallel – which implies that it is also planar – is $\mathcal{L}$-complete.

For $\mathcal{L}$ several simple graph problems are known to be complete with respect to $\mathcal{AC}^0$-reductions: for example, whether a graph is a forest or even a tree, or whether in a given forest G two nodes belong to the same tree (for a list of complete problems see [9, 14]). In this paper we will prove three problems for series-parallel graphs to be $\mathcal{L}$-complete: the *recognition problem,* the *reachability problem,* and *counting the number of paths mod 2.* While the hardness of these problems can be obtained in a straightforward way, it requires a lot of algorithmic effort to prove that the lower bound can actually be achieved. Thus, the main technical contribution of this paper are new graph-theoretical notions and algorithmic methods that allow us to solve these problems using only logarithmic space.

Furthermore, not only decision problems for series-parallel graphs turn out to be tractable. A decomposition of such graphs can be computed within the same space bound as well. For general graphs *counting the number of paths* is one of the generic complete problems for the class $\#\mathcal{L}$ [2]. Thus, this problem is not computable in $\mathcal{FL}$, the functional deterministic Logspace complexity class, unless certain hierarchies collapse. We will prove that restricting to series-parallel graphs the counting problem can be solved in $\mathcal{FL}$. This will be achieved by combining our space effi cient reachability decision procedure with a modular representation of numbers requiring only little space, and the recent result that a Chinese Remainder Representation can be converted to the standard binary one in logarithmic space [8].

Because of the relation between $\mathcal{L}$ and parallel time complexity classes defi ned by the EREW PRAM model (see [15]) these new algorithms can be modifi ed to solve these problems in logarithmic time on EREW PRAMs as well. Finally, these results can also be extended to generalizations of series-parallel graphs: multiple source or multiple sink, and minimal vertex-series-parallel graphs.

This paper is organized as follows. In Section 2 we will prove the $\mathcal{L}$-hardness of the reachability and the recognition problem. Procedures solving these problems within logarithmic space will be described in detail in Section 3. Section 4 outlines an algorithm that generates an edges-ordering that reflects the structure of a given series-parallel graph. Based on this ordering we sketch a decomposition algorithm in Section 5. In Section 6, we combine the methods presented so far to solve the path counting problem. Finally, in Section 7 and 8 it will be indicated how these results can be extended to generalizations of series-parallel graphs. We finish with some conclusions and open problems.

2 Hardness Results

To establish meaningful lower bounds for the deterministic space complexity class $\mathcal{L}$ one has to restrict the concept of polynomial time many-one reductions to simpler functions. We consider the usual requirement that the reducing function f can be computed in $\mathcal{AC}^0$. The $\mathcal{L}$-hardness for series-parallel graphs can be shown in a direct way.

$\mathcal{AC}^0$-many-one reductions are defined as follows. Let Σ be an alphabet. For problems $A, B \subseteq \Sigma^*$ $A \leq_m^{\mathcal{AC}^0} B$ holds if there exists a function f that is computable by unbounded fanin circuits of polynomial size and constant depth with the property that for all $x \in \Sigma^*$ holds: $x \in A$ iff $f(x) \in B$.

Theorem 1 *The following problems are hard for $\mathcal{L}$ under $\mathcal{AC}^0$ reducibility:*

1. *reachability in series-parallel graphs,*
2. *recognition of series-parallel graphs, and*
3. *counting the number of paths mod 2.*

Proof: Let L be a language in $\mathcal{L}$ and M a logarithmic space-bounded deterministic Turing machine accepting L by taking at most n^k steps on inputs X of length n, where k is a fixed exponent. We may assume that M has unique final configurations C_{acc}, the accepting one, and C_{rej}, the rejecting one. In addition, all configurations C of M on X are time-stamped, that means are actually tuples (C, t) with $0 \leq t \leq n^k$. Then the successor configuration of (C, t) is $(C', t+1)$ if $t < n^k$ and M can move in one step from C to C'. If C is a final configuration and $t < n^k$ then $(C, t+1)$ is the successor of (C, t). For input X we construct a directed graph G_X, where the time-stamped configurations (C, t) are the vertices of G_X and edges represent (the inverse of) the successor relation: G_X contains the edge $((C', t+1), (C, t))$ iff $(C', t+1)$ is a successor of (C, t). Obviously, G_X is a forest consisting of trees with roots of the form (C, n^k).

To prove the hardness of the *reachability problem* we augment G_X by two new nodes u and v. For every configuration (C, n^k) the edge $(u, (C, n^k))$ is added, and for every leaf (C, t) the edge $((C, t), v)$. It is easy to see that the resulting graph is series-parallel with source u and sink v. Furthermore, it contains a path from (C_{acc}, n^k) to $(C_{init}, 0)$, where C_{init} represents the starting configuration of M iff M accepts X. The reduction itself can be computed in $\mathcal{AC}^0$. It is illustrated in Figure 1.

In order to prove the hardness of the *recognition problem* we modify G_X as follows. For every configuration $C \notin \{C_{acc}, C_{rej}\}$ add an edge $((C_{acc}, n^k), (C, n^k))$ to G_X. Also add the edge $((C_{init}, 0), (C_{rej}, n^k))$. Augment the graph by a new node v, and for every leaf

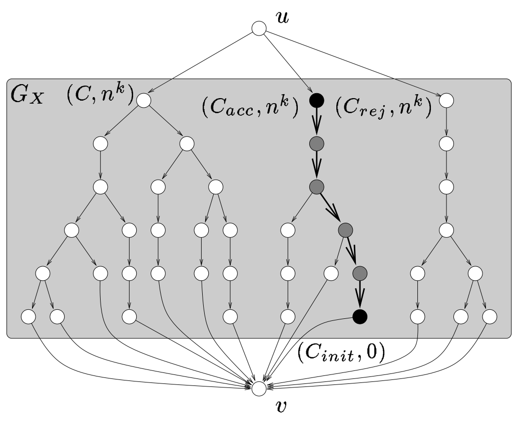


Figure 1: The $\mathcal{L}$-hardness reduction to reachability problem for series-parallel graphs. The bold path from (C_{acc}, n^k) to $(C_{init}, 0)$ represents an accepting computation of M on X in reverse direction.

$(C, t) \neq (C_{init}, 0)$ add the edge $((C, t), v)$. If M accepts X then the new graph is series-parallel with source (C_{acc}, n^k) and sink v. Otherwise, it contains a cycle connecting $(C_{init}, 0)$ and (C_{rej}, n^k).

Finally, let us consider the problem to *count the number of paths mod 2.* To prove the hardness we will modify again the computation forest G_X. Let

$$G_{\text{dup}} := (\{a, b, c, d\}, \{(a, b), (a, c), (b, d), (c, d)\}) \ .$$

To each leaf $(C, k) \neq (C_{init}, 0)$ and to each root $(C, t) \neq (C_{acc}, n^k)$ of G_X attach a separate copy of G_{dup} drawing edges $((C, k), a)$, resp. $(d, (C, t))$. Then we add new nodes u, v and connect u with each source u' of the modified graph by (u, u'), and each sink v' of the graph with v by (v', v). Clearly, the resulting graph is series-parallel. Furthermore, the number of u-v paths is odd iff there exists a path from a node (C_{acc}, n^k) to $(C_{init}, 0)$, i.e. M accepts X. ∎

3 Recognition and Reachability in Logspace

Establishing corresponding upper bounds is not obvious at all. We will give a space efficient characterization of series-parallel graphs by forbidden subgraphs and exploit the structure of internal paths very thoroughly. Assume that the nodes of the input graph G are represented by the set of numbers $\{1, 2, \ldots, n\}$. G is given by a list of edges $(i_1, j_1), (i_2, j_2), \ldots (i_m, j_m)$, where i_k, j_k are binary representations of the names of the nodes. Let **pred**(v) denote the set of immediate predecessors of v, and **pred**(v, i) the i-th immediate predecessor of v according to the ordering implicitly given by the specification of G. Similarly, let **succ**(v) and **succ**(v, i) be the set of immediate successors of v, resp. its i-th immediate successor. succ(v, i) and pred(v, i) can be computed in deterministic logarithmic space: the Turing machine searches through the list of edges looking for the i-th entry that starts (resp. ends) with v.

Define **pred**$^+(v)$, resp. **succ**$^+(v)$, as the transitive closure of pred(v), resp. succ(v), not containing v, and **pred**$^*(v)$:= pred$^+(v) \cup \{v\}$ and **succ**$^*(v)$:= succ$^+(v) \cup \{v\}$. To shorten the notation, let us introduce the predicate **PATH(u, v)** being true iff the given graph G possesses a path from node u to v. Thus,

$$\mathsf{PATH}(u,v) \quad \Longleftrightarrow \quad u \in \text{pred}^*(v) \quad \Longleftrightarrow \quad v \in \text{succ}^*(u) \; .$$

Remember that deciding PATH for arbitrary graphs is $\mathcal{NL}$-complete. To construct a deterministic space efficient algorithm solving this problem for series-parallel graphs we introduce the following concepts:

lm-down(v)	:=	the maximal acyclic path $v = u_1, u_2, \ldots, u_l$ with $u_{i+1} = \text{succ}(u_i, 1)$,
lm-up(v)	:=	the maximal acyclic path $v = u_1, u_2, \ldots, u_l$ with $u_{i+1} = \text{pred}(u_i, 1)$,
lm-pred*(v)	:=	$\{u \mid \text{lm-down}(u) \cap \text{lm-up}(v) \neq \emptyset\}$, and
lm-succ*(v)	:=	$\{u \mid \text{lm-down}(v) \cap \text{lm-up}(u) \neq \emptyset\}$.

Here, "lm" stands for *left-most*, that means in each node u_i the path follows the first edge as specified by the representation of G. A path being acyclic requires that all its nodes u_i are different. Thus, a maximal acyclic down-path either ends in a sink or stops immediately before hitting a node as the left-most successor a second time. These concepts are illustrated in Fig. 2 and 3

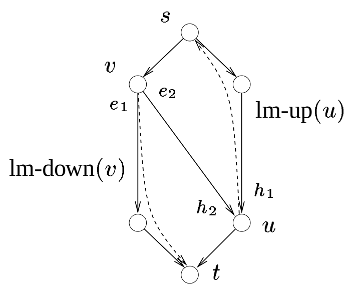


Figure 2: $u \notin$ lm-down(v) and $v \notin$ lm-up(u).

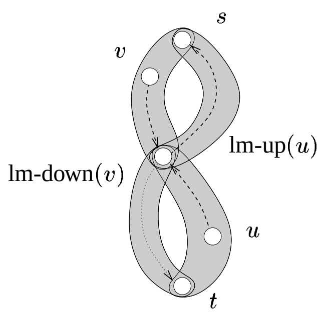


Figure 3: $v \in$ lm-pred$^*(u)$ and $u \in$ lm-succ$^*(v)$.

The sets can be decided by the procedure membership-lm-down(u, v) that returns TRUE if and only if $u \in$ lm-down(v).

```
procedure membership-lm-down$(u, v)$
  let $n$ be the number of nodes in $G$;
  $x := v$;
  $i := 1$;
  while $x \neq u$ and $|\text{succ}(x)| > 0$ and $i \leq n$ do
    let $x := \text{succ}(x, 1)$; $i := i + 1$ od
  if $x = u$ then return TRUE else return FALSE
```

Testing whether $u \in$ lm-up(v) can be done by the algorithm which is just the symmetric dual hence we omit a description of a corresponding procedure membership-lm-up(u, v).

To check if $u \in$ lm-pred*(v) one can use the procedure membership-lm-pred(u, v), which uses membership-lm-down and membership-lm-up to decide lm-down and lm-up.

```
procedure membership-lm-pred($u, v$)
  $result :=$ FALSE
  forall nodes $x$ in $G$ do
    if $x \in$ lm-down($u$) and $x \in$ lm-up($v$) then let $result :=$ TRUE od
  return $result$
```

In the dual way we can test whether $u \in$ lm-succ*(v). Hence it follows:

Lemma 1 *For an arbitrary graph G and node v the membership problem for the sets* lm-down(v), lm-up(v), lm-pred*(v), *and* lm-succ*(v) *can be solved deterministically in logarithmic space.*

Definition 2 *A graph G is called* ***st*-connected** *if G is has a unique source named s and a unique sink named t, and for every node v it holds:* PATH(s, v) *and* PATH(v, t), *i.e. there exists a path from s to v and a path from v to t.*

We start with a procedure called preliminary-test. For an acyclic graph G it returns TRUE iff G is st-connected. If G contains a cycle $C = (v_1, v_2, \ldots, v_l)$ the procedure will detect the cycle if it is on a left-most path. In such a case the procedure outputs FALSE. Cycles that are not of this form will not be detected, and the procedure erroneously may output TRUE.

```
procedure preliminary-test($G$)
  if not [$G$ has a unique source $s$ and a unique sink $t$]
    then return FALSE and exit
  forall nodes $v$ in $G$ do
    if $t \notin$ lm-down($v$) or $s \notin$ lm-up($v$) then return FALSE and exit
  return TRUE
```

Lemma 2 *The procedure* preliminary-test *can be implemented deterministically in Logspace. Moreover, if* preliminary-test(G) *outputs* TRUE *then G is st-connected. If it outputs* FALSE *then at least one of the following conditions holds:*

1. *G has more then one source or more than one sink, or*
2. *G is st-connected, but it has a cycle.*

The proof of this lemma is straightforward and we omit it. Note that a graph G with output TRUE can still have a cycle. To detect this property is diffi cult for deterministic machines since this question can easily be shown to be $\mathcal{NL}$-complete. Therefore, we look for a simpler task.

Let W denote the graph shown in Fig. 4. A graph W' is *homeomorphic* to W if it contains four distinct vertices a, b, c, d and pairwise internally vertex disjoint paths $P_{ab}, P_{ac}, P_{bd}, P_{cd}$ and P_{bc}. If G contains a homeomorphic image of W as a subgraph then W is called a *minor* of G. The following characterization of series-parallel graphs by forbidden minors has been known for long.

Theorem 2 ([10], [17]) *Let G be an st-connected acyclic graph. Then G is series-parallel iff W is not a minor of G.*

To make series-parallel graph recogniton space efficient, instead of searching for the forbidden minor W we will use the following characterization. Let H be a graph with four distinct nodes z_1, z_2, z_3, z_4 such that

1. $(z_1, z_2), (z_3, z_4)$ are edges of H and $\mathsf{PATH}(z_1, z_4)$,
2. $\neg\ \mathsf{PATH}(z_1, z_3)$ and $\neg\ \mathsf{PATH}(z_2, z_4)$.

These conditions are illustrated in Fig. 5. In the following we will show how H can be used to determine whether a graph is series-parallel. We say that H is an *induced subgraph* of G if G contains four nodes z_1, z_2, z_3, z_4 which fulfil these connectivity conditions.

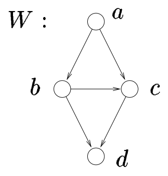


Figure 4: The forbidden minor W.

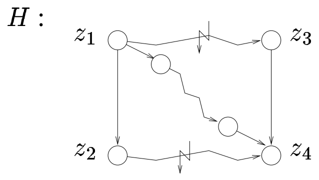


Figure 5: The forbidden induced subgraph H.

Theorem 3 *Let G be an st-connected acyclic graph. Then G is series-parallel iff it does not contain H as an induced subgraph.*

Proof: Let G be acyclic and st-connected with source s and sink t. We will show that G contains W as minor iff it contains H as an induced subgraph.

First assume that W is a minor and denote the corresponding nodes of G by a, b, c, d. Because of the acyclicity G cannot contain a path from d to c. Hence, on the path P_{bd} there must be an edge (b_1, b_2) such that

$$\mathsf{PATH}(b_1, c) \qquad \text{and} \qquad \neg\ \mathsf{PATH}(b_2, c)\ .$$

Similarly, on the path P_{ac} one can find an edge (c_1, c_2) such that

$$\mathsf{PATH}(b_1, c_2) \qquad \text{and} \qquad \neg\ \mathsf{PATH}(b_1, c_1)\ .$$

$\neg\ \mathsf{PATH}(b_2, c)$ implies $\neg\ \mathsf{PATH}(b_2, c_2)$. Hence, the nodes $z_1 := b_1$, $z_2 := b_2$, $z_3 := c_1$, and $z_4 := c_2$ fulfill the conditions from above.

To prove the opposite implication assume that nodes z_1, z_2, z_3, z_4 form the basis of H as an induced subgraph of G. Choose $b := z_1$ and $c := z_4$. Moreover, let a be the first common predecessor of b and z_3 according to left-most paths from b and z_3 back to the source of G. Since G is st-connected such a node a exists. Hence, we have found internally vertex disjoint paths P_{ab} and P_{az_3} from a to b and from a to z_3, respectively. Analogously, let d be the first common successor of z_2 and c according to the left-most path order which gives rise to vertex disjoint paths P_{z_2d} and P_{cd}. The acyclicity of G and $\neg\ \mathsf{PATH}(z_1, z_3)$ and $\neg\ \mathsf{PATH}(z_2, z_4)$

imply that the path P_{ab}, the path $P_{ac} = P_{az_3} \circ c$ (here '$\circ$' means a concatenation), the path $P_{bd} = b \circ P_{z_2 d}$, and the path P_{cd} are also internally vertex disjoint from the path $P_{bc} = P_{z_1 z_4}$, in other words W is a minor of G. ∎

Now, we will deduce the key property that makes reachability in SP-graphs easier compared to arbitrary graphs. Although the parallel composition operator introduces a lot of nondeterminism into the structure of these graphs when trying to find a path from a node u to a node v this question can be solved by considering the unique lm-down-path starting at u and the unique lm-up-path starting in v and deciding whether these two intersect. In other words, it holds:

Theorem 4 *If G is series-parallel then* pred*(v) = lm-pred*(v) *for every node v.*

Proof: Assume that pred*$(v) \neq$ lm-pred*(v) for some node v of G. We will show that then H has to be an induced subgraph of G – a contradiction to Theorem 3.

Obviously, v cannot be the source s of G. Since G is st-connected and acyclic every lm-down-path from an arbitrary node u has to terminate in the sink t. Thus, for t holds pred*(t) = lm-pred*$(t) = V$, and hence $v \neq t$.

Let $u_1 \in$ pred*$(v) \setminus$ lm-pred*(v). Since every lm-up-path terminates in the source s we can conclude $u_1 \neq s$. Let $u_1, \ldots, u_k = t$ be the leftmost down-path lm-down(u_1). $u_1 \notin$ lm-pred*(v) implies that $u_i \neq v$ for all $i \in [1..k]$. Furthermore, let $v = v_1, v_2, \ldots, v_\ell = s$ be the leftmost up-path lm-up(v) from v.

Since $u_1 \in$ pred*(v_1) there exists a non-trivial path from u_1 to v_1. On the other hand, because of $v_\ell = s$ and $v_1 = v \neq s$ it holds $\neg$ PATH(v_1, v_ℓ) , and similarly because of $u_k = t$ and $v_1 \neq t$, $\neg$ PATH(u_k, v_1). Hence, there exist $i \in [1..k-1]$ and $j \in [1..\ell-1]$ such that PATH(u_i, v_j), $\neg$ PATH(u_i, v_{j+1}), and $\neg$ PATH(u_{i+1}, v_j).

The nodes $z_1 := u_i$, $z_2 := u_{i+1}$, $z_3 := v_{j+1}$, and $z_4 := v_j$ prove that H is an induced subgraph of G. ∎

Thus, if for some node v of G the relation pred*(v) = lm-pred*(v) is violated one can conclude that G is not series-parallel. This equality, however, can be tested space efficiently.

Lemma 3 *There exists a deterministic logarithmic space-bounded Turing machine that for arbitrary $v \in G$ decides whether* pred*(v)=lm-pred*(v).

Proof: Assume that pred*$(v) \neq$ lm-pred*(v). First, we claim that there has to be an edge $(u, w) \in E$ such that $u \in$ pred*$(v) \setminus$ lm-pred*(v) and $w \in$ lm-pred*(v). To see this, let $x \in$ pred*$(v) \setminus$ lm-pred*(v) and $x = u_1, u_2, u_3, \ldots, u_k = v$ be a down-path from x to v. Obviously, $u_1, u_2, \ldots, u_k \in$ pred*(v), $u_1 \notin$ lm-pred*(v), and $u_k \in$ lm-pred*(v). Therefore, there exists an index $i \in [1..k-1]$ such that $u_i \in$ pred*$(v) \setminus$ lm-pred*(v) and $u_{i+1} \in$ lm-pred*(v). This proves our claim. Now it is easy to see that the following algorithm answers the question whether pred*(v)=lm-pred*(v):

```
procedure equality-test($v$)
  $result$ := TRUE
  forall edges $(u, w)$ in $G$ do
    if $(u \notin$ lm-pred*$(v))$ $\wedge$ $(w \in$ lm-pred*$(v))$ then $result$ := FALSE od
  return $result$
```
∎

Corollary 1 *Let G be an st-connected graph with* $\text{pred}^*(v) = \text{lm-pred}^*(v)$ *for every node v. Then reachability within G can be decided in $\mathcal{L}$.*

Proof: The identity $\text{pred}^*(v) = \text{lm-pred}^*(v)$ guarantees that for all w in G holds:

$$v \text{ is reachable from } w \iff \texttt{PATH}(w, v) \iff w \in \text{pred}^*(v) \iff w \in \text{lm-pred}^*(v) .$$

Lemma 1 gives that the membership problem for $\text{lm-pred}^*(v)$ can be solved space effi ciently. ∎

From the corollary above and from Theorem 4 follows immediately:

Theorem 5 *The reachability problem in SP-graphs is in $\mathcal{L}$.*

Now, the following procedure SER-PAR decides for an arbitrary graph G whether it is series-parallel.

```
procedure  SER-PAR($G$)
 1  if preliminary-test($G$) returns FALSE then return FALSE and exit
 2  forall nodes $v$ in $G$ do
 3    if $\text{pred}^*(v) \neq \text{lm-pred}^*(v)$ then return FALSE and exit od
 4  forall pairs of nodes $x, y$ in $G$ do
 5    if $x \in \text{lm-pred}^*(y) \wedge y \in \text{lm-pred}^*(x)$
 6      then return FALSE and exit od
 7  forall pairs of edges $(z_1, z_2), (z_3, z_4)$, with $z_1 \neq z_4$ do
 8    if $z_1 \in \text{lm-pred}^*(z_4) \wedge z_1 \notin \text{lm-pred}^*(z_3) \wedge z_2 \notin \text{lm-pred}^*(z_4)$
 9      then return FALSE and exit od
10  return TRUE
```

The correctness of this procedure can be seen as follows. From Lemma 2 one can conclude that the algorithm stops at line 1 and outputs FALSE if G has more then one source or more than one sink, or G is st-connected, but it has a cycle. Hence, G is not series-parallel and the answer FALSE is correct. On the other hand, if the procedure does not stop at line 1 then G is st-connected.
If SER-PAR(G) outputs FALSE in line 3 then $\text{pred}^*(v) \neq \text{lm-pred}^*(v)$ for some node v. By Theorem 4 it follows that this answer is correct, too.
If the algorithm continues, we can presuppose at the beginning of line 4 that G is st-connected and for every v it holds $\text{pred}^*(v) = \text{lm-pred}^*(v)$. In lines 4-6 we check whether G is acyclic, and stop if not. The answer will be correct since $\text{lm-pred}^*(y)$ contains all predecessors of a node y.
Let us recapitulate the conditions a graph G has to fulfi ll such that SER-PAR(G) does not stop before line 7: G has to be st-connected, acyclic and for every pair of nodes x, y in G it holds: $\texttt{PATH}(y, x) \iff y \in \text{lm-pred}^*(x)$. This guarantees that in lines 7-9 the existence of H as an induced subgraph is tested correctly. Finally, since all tests applied can be performed in deterministic logarithmic space we can conclude:

Theorem 6 *The question whether a graph is series-parallel can be decided in $\mathcal{L}$.*

4 An Edge Ordering Algorithm

For a graph specified by a list of edges we have made no assumptions about their ordering. In particular, this ordering is not required to reflect the construction process of the SP-graph in any way. In this section we present a log-space algorithm that given an series-parallel SP-graph G outputs a special ordering called **SP-ordering**. The crucial property of this ordering is that for any series-parallel component C with source v all immediate successors of v in C are enumerated with consecutive integers. Speaking formally, for a node v the sequence **SP-succ(v)** is a permutation of $\mathrm{succ}(v)$ such that for all $u \in \mathrm{succ}^{+}(v)$ the set $\{\ i \mid \texttt{SP-succ}(v,i) \in \mathrm{pred}^{*}(u)\ \}$ consists of consecutive integers. Here, for $1 \le i \le |\mathrm{succ}(v)|$ the value $\texttt{SP-succ}(v,i)$ denotes the i'th vertex in the SP-ordering of $\mathrm{succ}(v)$. Recall that $\mathrm{succ}(v,i)$ is the i'th immediate successor of v according to the ordering implicitly given by the input specification. Hence, in general $\texttt{SP-succ}(v,i)$ will be different from $\mathrm{succ}(v,i)$. To compute the SP-ordering we will use $\mathrm{succ}(v,i)$ and the following functions for a node u:

$$\begin{array}{lrcl} \text{for } u \neq s, & \texttt{START}(u) & := & \text{closest } v \in \mathrm{pred}^{+}(u) \text{ such that every path from } s \text{ to } u \text{ contains } v, \\ \text{for } u \neq t, & \texttt{FINAL}(u) & := & \text{closest } v \in \mathrm{succ}^{+}(u) \text{ such that every path from } u \text{ to } t \text{ contains } v. \end{array}$$

For all $u \neq s$, $\texttt{START}(u)$ is well defined. It gives the source v of a smallest series-parallel subgraph of G that contains u and all its immediate predecessors. Analogously, $\texttt{FINAL}(u)$ for $u \neq t$ is the sink of a smallest series-parallel subgraph containing u and its immediate successors. If u has only one immediate predecessor v then obviously $\texttt{START}(u) = v$. Otherwise, $\texttt{START}(u)$ will be computed by finding the closest common predecessor of the left-most up-paths from any immediate predecessor of u to the source of G. $\texttt{FINAL}(u)$ can be computed in an analogous way. We define an inverse relation by

$$\begin{array}{rcl} \texttt{START}^{-1}(v) & := & \{u \mid \texttt{START}(u) = v\}\ , \\ \texttt{FINAL}^{-1}(v) & := & \{u \mid \texttt{FINAL}(u) = v\}\ . \end{array}$$

Observe that

$$\texttt{START}^{-1}(v) \subseteq \mathrm{succ}^{*}(v) \qquad \text{and} \qquad \texttt{FINAL}^{-1}(v) \subseteq \mathrm{pred}^{*}(v)\ .$$

The set $\texttt{START}(u)$ and $\texttt{FINAL}(u)$ can be computed in logarithmic space as follows:

```
procedure START($u$)
  if $u \neq s$ then $v := \mathrm{pred}(u,1)$
    for $i = 2$ to $|\mathrm{pred}(u)|$ do
      while $v \notin$ lm-up(pred($u,i$)) do
        $v := \mathrm{pred}(v,1)$ od od
    return $v$
  endif
```

```
procedure FINAL($u$)
  if $u \neq t$ then $v := \mathrm{succ}(u,1)$
    for $i = 2$ to $|\mathrm{succ}(u)|$ do
      while $v \notin$ lm-down(succ($u,i$)) do
        $v := \mathrm{succ}(v,1)$ od od
    return $v$
  endif
```

Using the above functions the membership problems for the sets $\texttt{START}^{-1}(v)$ and $\texttt{FINAL}^{-1}(v)$ can be decided in logarithmic space. Therefore we conclude:

Lemma 4 *Let G be a series-parallel graph, then for every node u in G the values $\texttt{START}(u)$, $\texttt{FINAL}(u)$, $\texttt{START}^{-1}(u)$, and $\texttt{FINAL}^{-1}(u)$ can be computed in logarithmic space.*

Proof: It remains to show that the procedures START(u) and FINAL(u) compute the correct values. By definition, $\texttt{START}(u)$ is the first common node v of all up-paths starting from nodes in pred(u). It is easy to see that for SP-graphs v is at the same tame the first common node of all *leftmost* up-paths starting from nodes in pred(u). On the other hand, the specification of the procedure START implies that it returns such a node v. The argument for FINAL is analogous. ∎

Next, we will introduce a notion that is motivated by the analysis of SP-graphs above.

Definition 3 *The set of* **bridge nodes** *between two nodes v_1 and v_2 is defined as follows:*

$$\begin{aligned}\texttt{BRIDGES}(v_1, v_2) \quad := \quad & \{u \in \texttt{START}^{-1}(v_1) \cap \text{pred}^{+}(v_2) \mid \\ & \qquad \forall w \in \text{succ}^{+}(u) \cap \text{pred}^{+}(v_2) \ : \ w \notin \texttt{START}^{-1}(v_1)\} \ .\end{aligned}$$

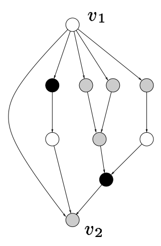


Figure 6: The nodes in $\texttt{START}^{-1}(v_1)$ are marked darker;
in black: the subset $\texttt{BRIDGES}(v_1, v_2)$.

Less formally speaking, to find the set of bridges between two nodes v_1, v_2 consider the induced subgraph $\boldsymbol{G_{v_1,v_2}}$ of G that has v_1 as source and v_2 as sink and consists of the nodes in succ$^*(v_1) \cap$ pred$^*(v_2)$. Then on every non-trivial path in G_{v_1,v_2} from v_1 to v_2 we select those node u that is closest to v_2 and for which v_1 is the $\texttt{START}$-node.

Lemma 5

1. *If v_2 is a successor of v_1 then the subgraph G_{v_1,v_2} is a nonempty SP-graph.*

2. *If G_{v_1,v_2} has at least 1 additional node besides its source v_1 and sink v_2 then it holds $\texttt{START}^{-1}(v_1) \cap \text{pred}^{+}(v_2) \neq \emptyset$ and $\texttt{BRIDGES}(v_1, v_2)$ contains some elements, namely at least those $u \in \texttt{START}^{-1}(v_1) \cap \text{pred}^{+}(v_2)$ that have no successor in this set.*

3. *Every path from v_1 to v_2 different from the directed edge (v_1, v_2) contains a node that belongs to $\texttt{BRIDGES}(v_1, v_2)$.*

4. *Bridge nodes are pairwise incomparable, i.e. none is a successor of another.*

5. *If v_2 is not a successor of v_1 then $\texttt{BRIDGES}(v_1, v_2) = \emptyset$.*

Proof: The first property is obvious using the characterization by the forbidden minor W.
The second property follows from the observation that for an SP-graph with source v_1, sink v_2 and at least 1 additional node, $\texttt{START}^{-1}(v_1)$ always contains another element in addition to v_2.

For claim 3, assume that $P = v_1, u_1, u_2, \ldots, u_m, v_2$ is a path of length at least 2 from v_1 to v_2 that does not intersect $\texttt{BRIDGES}(v_1, v_2)$. By definition, $u_1 \in \texttt{START}^{-1}(v_1)$. Let u_ℓ be the last node before v_2 of P that belongs to $\texttt{START}^{-1}(v_1)$. If $u_\ell \notin \texttt{BRIDGES}(v_1, v_2)$ there must exists $w \in (\text{succ}^+(u_\ell) \cap \text{pred}^+(v_2) \cap \texttt{START}^{-1}(v_1)) \setminus P$ that prevents u_ℓ from becoming a bridge node. Furthermore, there has to be a path from v_1 to w that does not contain u_ℓ. Now let $s' \neq u_\ell$ be the nearest common predecessor of u_ℓ and w, and $t' \neq w$ be the nearest common successor of u_ℓ and w. Then the nodes s', u, w, t' span W as a minor of G.

The last two properties follow easily from the definition. ∎

All sets used in the definition of $\texttt{BRIDGES}$ can be decided in logarithmic space, thus

Lemma 6 *For an SP-graph the set* $\texttt{BRIDGES}(v_1, v_2)$ *can be computed in logarithmic space for arbitrary nodes* v_1, v_2.

The sets $\texttt{BRIDGES}$ will be used to decompose a given series-parallel graph G. We will order the elements u in $\texttt{BRIDGES}(v_1, v_2)$ according to their names. For $1 \leq i \leq |\texttt{BRIDGES}(v_1, v_2)|$ let $\texttt{BRIDGES}(v_1, v_2, i)$ be the i-th largest element according to this ordering. By enumerating all nodes of G and testing for membership in $\texttt{BRIDGES}(v_1, v_2)$ the node $\texttt{BRIDGES}(v_1, v_2, i)$ can be computed in logarithmic space as well.

Lemma 7 *For an SP-graph G with nonempty set* $\texttt{BRIDGES}(s, t)$ *of cardinality k let u_i denote the i-th element in this set, and let G_i be the series composition of G_{s,u_i} and $G_{u_i,t}$. Then it holds:*

1. *if G has an edge (s, t) then G is the parallel composition of $G_0 := \{(s, t)_1, \ldots, (s, t)_\ell\}$ and $G_1, G_2, \ldots, G_k$,*
2. *if $(s, t) \notin E$ and $k = 1$ then G is the series composition of G_{s,u_1} and $G_{u_1,t}$,*
3. *if $(s, t) \notin E$ and $k > 1$ then G is the parallel composition of $G_1, G_2, \ldots, G_k$.*

Proof: First let us consider the case $(s, t) \notin E$.
If $k = 1$ then it remains to show that there can be no nodes outside of $G_{s,u_1} \cup G_{u_1,t}$ and no edges connecting G_{s,u_1} with $G_{u_1,t}$. These two properties follow easily from Lemma 5(3) since otherwise one could construct a path from s to t that avoids the only bridge node u_1.

For $k > 1$, we have to prove for arbitrary $1 \leq i < j \leq k$:

(a) $\text{pred}^*(u_i) \cap \text{pred}^*(u_j) = \{s\}$,

(b) $\text{succ}^*(u_i) \cap \text{succ}^*(u_j) = \{t\}$,

(c) $E \cap (\text{pred}^*(u_i) \times \text{succ}^*(u_j)) = \emptyset$,

(d) $\forall\, v \in V \quad \exists\, \ell \leq k \quad v \in G_{s,u_\ell} \cup G_{u_\ell,t}$.

To show the first property let us assume to the contrary that $v \neq s$ is a common predecessor of both u_i and u_j. Let v be the closest node with this property. Since $\mathtt{START}(u_i) = s$ there exist three pairwise internally vertex disjoint paths: $P_{s,v}$, P_{v,u_i} and P_{s,u_i}. Moreover, there has to be a path P_{v,u_j} that is internally disjoint with the previous paths. Finally, let t' be the nearest common successor of u_i and u_j. Then s, u_i, v, t' would span the forbidden minor W.

For claim (b) assume to the contrary that v is a nearest common successor of u_i and u_j different from t. By (a) every pair of paths P_{s,u_i} and P_{s,u_j} is internally vertex disjoint. The same property holds for paths $P_{u_i,v}$ and $P_{u_j,v}$ because of the choice of v. The concatenations of P_{s,u_i} with $P_{u_i,v}$ and P_{s,u_j} with $P_{u_j,v}$ yields two disjoint paths from s to v which would imply $v \in \mathtt{START}^{-1}(s)$ or the existence of the forbidden minor W. But this is a contradiction to u_i, u_j being bridge nodes.

Assume now that there exists an edge $(v, w) \in E$ with $v \in \text{pred}^*(u_i)$ and $w \in \text{succ}^*(u_j)$. From (a) and (b) we know that $v \neq u_i$ and $w \neq u_j$. Therefore, there exist a path $P_{s,v}$ from s to v that does not intersect $\mathtt{BRIDGES}(s, t)$ and similarly a path $P_{w,t}$ from w to t. The concatenation $P_{s,v} \circ w \circ P_{w,t}$ gives a path that would not intersect $\mathtt{BRIDGES}(s, t)$ – a contraction to Lemma 5(3).

For claim (d) assume that a node v of G does not belong to any of the subsets $\text{pred}^*(u_i) \cup \text{succ}^*(u_i)$. Since v has to be connected to s and t somehow there must be a path from s to t containing v which does not go through $\mathtt{BRIDGES}(s, t)$. This, however, would contradict Lemma 5(3).

Finally, the case $(s, t) \in E$ can be reduced to the other two cases by considering the component $G_0 := \{(s, t)_1, \ldots, (s, t)_\ell\}$ separately and then removing the edges $(s, t)_1, \ldots, (s, t)_\ell$ from G. ∎

Now we can specify the algorithm to compute an SP-ordering. The recursive procedure SP-sequence(u, t) outputs the sequence of immediate successors of a node u in an SP-ordering.

```
procedure SP-sequence$(u, v)$
  if $(u, v) \in E$ then output $v$ endif
  for $i = 1$ to $|\mathtt{BRIDGES}(u, v)|$ do
    SP-sequence$(u, \mathtt{BRIDGES}(u, v, i))$
```

Theorem 7 *For every node u of a series-parallel graph an SP-ordering of its immediate successors of u can be computed in logarithmic space.*

Proof: The correctness of the procedure SP-sequence follows directly from Lemma 7. A logarithmic space bound, however, will not be sufficient for this recursive approach. Instead, consider the following iterative implementation of SP-sequence(u, v) that recomputes the parameters of a recursive call.

```
 1  if $(u,v) \in E$ then output $v$ endif
 2  $i := 1$
 3  if $i > |\mathtt{BRIDGES}(u,v)|$ then goto 7
 4  $v := \mathtt{BRIDGES}(u,v,i)$;
 5  if $(u,v) \in E$ then output $v$ endif
 6  goto 2
 7  if $v = \mathtt{FINAL}(u)$ then exit endif
 8  compute $w \in \mathtt{START}^{-1}(u)$ such that $v \in \mathtt{BRIDGES}(u,w)$
 9  compute $i$ such that $\mathtt{BRIDGES}(u,w,i) = v$
10  $v := w$; $i := i + 1$;
11  goto 3.
```

This procedure uses only 4 variables: 3 to index the nodes of the input graph and 1 as a counter that is bounded by the maximal size of the sets $\mathtt{BRIDGES}(u,v)$. Hence, logarithmic space suffi ces. ∎

The SP-ordering of the predecessors of a node can be computed analogously.

5 The Decomposition in Logspace

A *decomposition tree* of a series-parallel graph provides information how this graph has been built using the parallel and serial constructors. It will also allow us to solve several counting problems for series-parallel graph.

Definition 4 *A binary tree $T = (V_T, E_T)$ with a labelling function $\sigma : V_T \to \{\mathtt{p}, \mathtt{s}\} \cup E$ is a* **decomposition tree** *of an SP-graph $G = (V, E)$ iff*

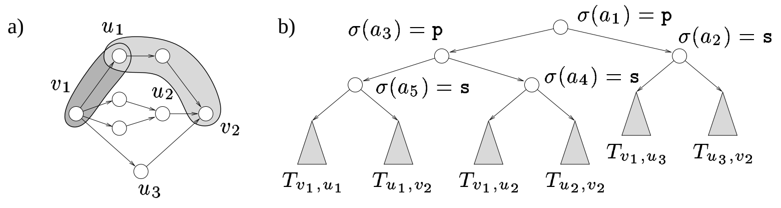
1. *leaves of T are labelled with elements of E, internal nodes with* p *or* s*,*
2. *G can be generated recursively using T as follows:*
   (a) *If T is a single node v then G consists of the single edge $\sigma(v)$ and its associated nodes;*
   (b) *otherwise, let T_1 and T_2 be the right (resp. left) subtree of T and G_i be SP-graphs with decomposition tree T_i:*
   if $\sigma(v) = \mathtt{p}$ then G is the parallel composition of G_1 and G_2,
   if $\sigma(v) = \mathtt{s}$ then G is the series composition of G_1 and G_2.

Note that the decomposition tree in general is not unique since several parallel compositions may be combined in different ways, and similarly several series compositions. However, if in a decomposition tree neighbouring nodes with the same label are collapsed to a single node such that the resulting tree may have nodes of outdegree larger than 2, but on every path from a leaf to the root the sequence of labels alternates between p and s the tree becomes unique.

Bridge nodes play an essential role when computing a decomposition tree. Given a series-parallel graph G with source s and sink t, the procedure SP-DECOMP(s,t) outputs the root DTR of a decomposition tree of G. To generate the nodes of the tree we use the procedure getnode(r) that allocates a needed amount of memory for a new node and assigns a pointer to this node to r. The decomposition algorithm is based on the characterization given in Lemma 7. For an example of such a decomposition see Fig. 7.

```
procedure SP-DECOMP$(u, v)$

if $|$BRIDGES$(u, v)| \geq 1$ then do
    getnode$(r)$; $\sigma(r)$ :=s;
    left$(r)$ := SP-DECOMP$(u,$ BRIDGES$(u, v, 1))$
    right$(r)$ := SP-DECOMP(BRIDGES$(u, v, 1), v)$
    DTR:=r;
    for $i := 2$ to $|$BRIDGES$(u, v)|$ do
        getnode$(c)$; $\sigma(c)$ :=s;
        left$(c)$ := SP-DECOMP$(u,$ BRIDGES$(u, v, i))$
        right$(c)$ := SP-DECOMP(BRIDGES$(u, v, i), v)$
        getnode$(b)$; $\sigma(b)$ :=p; left$(b)$ := DTR; right$(b)$ := $c$;
        DTR := $b$
    endfor
endif
if $E$ contains $k \geq 1$ copies $(u, v)_1, \ldots, (u, v)_k$ of $(u, v)$
    then do
    getnode(c); $\sigma(c) := (u, v)_1$;
    for $\ell = 2, \ldots, k$ do
        getnode(a); $\sigma(a) := (u, v)_\ell$;
        getnode(b); $\sigma(b)$ := p; left$(b) := a$; right$(b) := c$;
        $c := b$;
    endfor
    if $|$BRIDGES$(u, v)| \geq 1$ then do
            getnode$(b)$; $\sigma(b)$ :=p; left$(b)$ := DTR; right$(b)$ := $c$;
            DTR := $b$ od
      else DTR := $c$
    endif
endif
return DTR.
```



a)
u_1
v_1
u_2
v_2
u_3
b)
$\sigma(a_3) = \mathtt{p}$
$\sigma(a_1) = \mathtt{p}$
$\sigma(a_2) = \mathtt{s}$
$\sigma(a_5) = \mathtt{s}$
$\sigma(a_4) = \mathtt{s}$
T_{v_1,u_3}
T_{u_3,v_2}
T_{v_1,u_1}
T_{u_1,v_2}
T_{v_1,u_2}
T_{u_2,v_2}

Figure 7: An example of a decomposition tree generated by SP-DECOMP(v_1, v_2): $T_{v,u}$ are subtrees generated by SP-DECOMP that consist of a single edge or a larger component.

Theorem 8 *A decomposition tree of an SP-graph can be computed in $\mathcal{FL}$.*

Proof: It remains to estimate the space required by SP-DECOMP. To be efficient we do not store the values of the variables for each recursive activation of SP-DECOMP as it is done in a standard implementation of recursion. In our special situation these values of the calling activation of SP-DECOMP can be recomputed by using the procedure BACK-DECOMP(x, y) when returning from a recursive call.

```
procedure BACK-DECOMP$(x, y)$
1  $u :=$ source$(G)$;  $v :=$ sink$(G)$; $i := 1$
2  if  $(x = u \ \wedge \ y = \texttt{BRIDGES}(u, v, i)) \ \vee \ (x = \texttt{BRIDGES}(u, v, i) \ \wedge \ y = v)$
3     then return  $(u, v, i)$
4  if  $x, y \in$ succ*$(u) \cap$ pred*$(\texttt{BRIDGES}(u, v, i))$
5     then  $v := \texttt{BRIDGES}(u, v, i)$;  $i := 1$; goto 2
6  if  $x, y \in$ succ*$(\texttt{BRIDGES}(u, v, i)) \cap$ pred*$(v)$
7     then  $u := \texttt{BRIDGES}(u, v, i)$;  $i := 1$; goto 2
8  $i := i + 1$;   goto 3.
```

The procedure BACK-DECOMP on input x, y returns a pair u, v and an integer i with the following properties: if SP-DECOMP is called recursively for nodes (x, y) then u, v and i are the current values for this recursive call in lines 3, 4, 8, 9. Thus one can implement SP-DECOMP without any stack for the recursion, and it follows easily that only logarithmic space is needed. ∎

6 Path Counting Problems

In this section we show that for series-parallel graphs the classical problem to count the number of paths can be solved in $\mathcal{FL}$. For general graphs counting the number of paths is not solvable in $\mathcal{FL}$ – unless certain hierarchies collapse – since this problem is one of the generic complete problems for the class $\#\mathcal{L}$ [2]. Speaking more precisely, let us define the function $\#\texttt{PATH}(\boldsymbol{a}, \boldsymbol{b})$ as the number of different paths from a to b in G.

Theorem 9 *Restricted to series-parallel graphs $\#\texttt{PATH}$ can be computed in $\mathcal{FL}$.*

Proof: Consider the subgraph $G_{a,b}$ of G induced by $V' :=$ succ*$(a) \cap$ pred*(b). It is either empty (and then $\#\texttt{PATH}(a, b) = 0$), or it is a series-parallel graph with source a and sink b. Furthermore, all paths from a to b in G occur in $G_{a,b}$ as well, thus the number of paths is identical. A simple induction shows that $\#\texttt{PATH}(a, b)$ can be bounded by 2^{n+m}. Using the reachability algorithm presented in Section 3 we can also decide in Logspace whether an arbitrary edge of G belongs to $G_{a,b}$.

Let $T = (V_T, E_T)$ be the decomposition tree of $G_{a,b}$ and $z \leq n + m$ be its size. We interpret the tree as an arithmetic expression as follows. Every leaf represents the integer 1. An internal node v of T labeled by $\sigma(v) = \texttt{s}$ (resp. $\texttt{p}$) defines a multiplication (resp. addition) of the expressions generated by its sons. One can check easily that the value ρ of the root of T equals $\#\texttt{PATH}(a, b)$.

Below we sketch how ρ can be computed in logarithmic space. Let $p_1 < p_2 < \ldots$ be the standard enumeration of primes. The prime number theorem implies

$$\prod_{p_i \le n+m} p_i \;=\; e^{(n+m)(1+o(1))} \;>\; \#\mathtt{PATH}(a,b)\;. \tag{1}$$

Using the log-space algorithm of [7] one can transform T into a binary tree T' of depth $O(\log z)$ representing an arithmetic expression with the same value ρ as T.

We evaluate $T' \bmod p_i$ using the algorithm in [4]. For $p_i \le n+m$ this algorithm works in space $O(\log z + \log(n+m)) \le O(\log n)$. By inequality (1), taking all $p_i \le n+m$ the values $\rho \mod p_i$ gives a *Chinese Remainder Representation* of ρ. Using the results of Chiu, Davida, Litow, resp. Hesse [8, 12] that such a representation can be converted to the ordinary binary representation in Logspace, fi nishes the proof. ▮

Using the hardness result shown in Section 2 it follows, that the problem to compute $\#\mathtt{PATH} \bmod 2$ is $\mathcal{L}$-complete. Using the techniques presented so far one can also solve other counting problems, like determining the size of $G_{a,b}$ in $\mathcal{FL}$.

7 Generalization to Multiple Terminal Series-Parallel Graphs

The class of series-parallel graphs can be extended by no longer requiring a unique source and a unique sink.

Definition 5 *The family of multiple source series-parallel graphs, MSSP-graphs for short, is obtained by adding the following constructor:*

(C) **In-Tree Composition***: given two graphs G_1, G_2 select a node $\hat{v}$ in G_1, identify it with the sink $v_{out,2}$ of G_2 and form the union of both graphs.*

In-tree compositions may be applied several times, but only after completion of all parallel and series compositions. As soon as a graph contains several sources the series and parallel constructors can no longer be used.

Multiple sink series-parallel graphs with a unique source, but possibly several sinks can be defi ned in an analogous way. In the following, we will restrict ourselves to the fi rst extension – dual results hold for the second class.

Unlike ordinary series-parallel graphs, the reachability problem for an MSSP-graphs G cannot be solved by simply following the leftmost paths, since (see for example Fig. 8 a) $\mathsf{PATH}(x,y)$ no longer implies $\text{lm-down}(x) \cap \text{lm-up}(y) \ne \emptyset$. To decide reachability a more sophisticated test is required. Defi ne

$$\mathsf{ELUDE}(v) \;:=\; \{\, u \mid \exists\, w_1, w_2 \in \text{succ}(u) \;:\; v \in \text{lm-down}(w_1) \setminus \text{lm-down}(w_2) \,\}\;.$$

It should be obvious that the membership problem for $\mathsf{ELUDE}(v)$ can be decided in $\mathcal{L}$.

Lemma 8 *For ordinary series-parallel graphs it holds* $\mathsf{ELUDE}(v) \subseteq \text{lm-up}(v)$.

Proof: Assume to the contrary that some $u \in \mathtt{ELUDE}(v)$ does not belong to lm-up(v). Then we get a contradiction to Theorem 2 as follows: Let $v = v_0, v_1, v_2, \ldots, v_\ell$ be the leftmost up-path of v and let $w_1, w_2 \in \text{succ}(u)$ be nodes such that $v \in \text{lm-down}(w_1) \setminus \text{lm-down}(w_2)$. By assumption, $u \neq v_i$ for all i. Let k be the first index with $v_k \in \text{pred}^+(u)$ and let j be the largest index such that v_j lies on the path lm-down(w_1). Moreover if the paths $v_j, v_{j+1}, \ldots, v_k$ and lm-down(w_2) are not disjoint then let $w := v_h$ be the common node of these two paths with the largest index h; otherwise let w be the nearest common node of lm-down(v_j) and lm-down(w_2). Then v_k, v_j, u and w induce the forbidden minor W. ∎

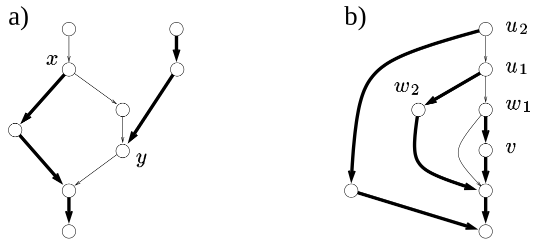


Figure 8: a) left-most paths that do not meet, b) computing $\mathtt{ELUDE}(v)$

Fig. 8 (a) shows that for MSSP-graphs the left-most up-paths may not contain $\mathtt{ELUDE}(y)$: the node x belongs to $\mathtt{ELUDE}(y)$, but lm-up(y) takes the other direction. But at least, almost the same proof as above shows:

Lemma 9 *The nodes in* $\mathtt{ELUDE}(v)$ *are linearly ordered, that is for all* $u \neq u' \in \mathtt{ELUDE}(v)$ *either* $u \in \text{succ}^+(u')$ *or* $u' \in \text{succ}^+(u)$.

Otherwise, one could find the forbidden minor W. Thus the closest predecessor of v in $\mathtt{ELUDE}(v)$, let us call it $\mathtt{NEXT\text{-}ELUDE}(v)$, is well defined if $\mathtt{ELUDE}(v) \neq \emptyset$.

Lemma 10 $\mathtt{NEXT\text{-}ELUDE}(v)$ *can be computed in $\mathcal{L}$.*

Proof: The node $\hat{u} = \mathtt{NEXT\text{-}ELUDE}(v)$ uniquely fulfils the property

$$\forall\, w \in \mathtt{ELUDE}(v) \setminus \{\hat{u}\} \qquad \hat{u} \in \bigcup_{x \in \text{succ}(w)} \text{lm-down}(x)$$

which can be tested space-efficiently. ∎

Now we define a set $\mathtt{CRITICAL}(v)$ of nodes that play the essential role for testing reachability in MSSP-graphs:

$$\mathtt{CRITICAL}(v) \quad := \quad \begin{cases} \{v\} \cup \mathtt{CRITICAL}(\mathtt{NEXT\text{-}ELUDE}(v)) & \text{if } \mathtt{ELUDE}(v) \neq \emptyset\,, \\ \{v\} & \text{else.} \end{cases}$$

The analogue of Theorem 4 looks as follows.

Theorem 10 *For every node v in an MSSP-graph G it holds*

$$\text{pred}^*(v) \;=\; \{u \mid \text{lm-down}(u) \cap \texttt{CRITICAL}(v) \neq \emptyset\}\ .$$

Proof: Let $w_0, w_1, \ldots, w_\ell$ denote the sequence of critical nodes for a given node v, i.e. $w_0 = v$, $w_{i+1} = \texttt{NEXT-ELUDE}(w_i)$, and $\texttt{ELUDE}(w_\ell) = \emptyset$. Assume to the contrary that for some $u \in \text{pred}^*(v)$ it holds: $\text{lm-down}(u) \cap \texttt{CRITICAL}(v) = \emptyset$. Let i be the maximal index such that $w_i \in \text{succ}^+(u)$. Then we define

$$A \;:=\; \text{succ}^*(u) \cap \text{pred}^*(w_i)\ .$$

Obviously, $w_i \in A$. By definition, $A \cap \texttt{ELUDE}(w_i) = \emptyset$, otherwise i would not be maximal. Let P_{u,w_i} be a path from u to w_i. Obviously $P_{u,w_i} \in A$. Because $w_{i+1} \notin \text{succ}^+(u)$ and $w_i \notin \text{lm-down}(u)$ there has to exist a node u' on the path P_{u,w_i} such that $w_i \notin \text{lm-down}(u')$ and either $w_i \in \text{succ}(u')$ or there exists a node $u'' \in \text{succ}(u')$, with $w_i \in \text{lm-down}(u'')$. Hence $u' \in A \cap \texttt{ELUDE}(w_i)$ – a contradiction. ∎

Combining this theorem together with Lemma 10 we obtain

Theorem 11 *The reachability problem for MSSP-graphs can be decided in $\mathcal{L}$.*

To verify whether a given graph is an MSSP one we can use the forbidden subgraph H again. However, a set of nodes fulfilling its PATH conditions can occur in MSSP-graph G, but only if z_3 belongs to another component G_2 of G than z_1 and z_2, which is connected to the rest graph of G via an in-tree composition step at z_4. Since this can also be verified in Logspace we obtain:

Theorem 12 *The recognition problem for MSSP-graphs is in $\mathcal{L}$.*

The notion of decomposition trees can be extended to this larger class of graphs as well.

Definition 6 *A binary tree $T = (V_T, E_T)$ with a labelling function $\sigma : V_T \to \{\texttt{p}, \texttt{s}, (\texttt{i} \times V)\} \cup E$ is a* **decomposition tree** *of an MSSP-graph $G = (V, E)$ iff*

1. *every leaf of T is labelled with an edge in E and every internal node with an element from $\{\texttt{p}, \texttt{s}, (\texttt{i} \times V)\}$;*
2. *G can be generated recursively using T as follows:*
   - *(a) If T is a single node v then G consists of the single edge $\sigma(v)$ and its associated nodes;*
   - *(b) otherwise, let T_1 (resp. T_2) be the right (resp. left) subtree of T and G_i be MSSP-graphs with decomposition tree T_i:*
     - *i. if $\sigma(v) = \texttt{p}$ then G is the parallel composition of G_1 and G_2,*
     - *ii. if $\sigma(v) = \texttt{s}$ then G is the series composition of G_1 and G_2,*

iii. if $\sigma(v) = (\mathtt{i}, v)$ then G is the in-tree composition of G_1 and G_2, where v is a node of G_1 and the sink of G_2 is identified with v.

To generate a decomposition tree for an MSSP-graph we first generate nodes that represent the in-tree composition steps, then the subtrees that describe the decomposition of the basic series-parallel graphs. To enumerate these subgraphs we will use the sources of the MSSP-graph. Note that there may arise ambiguities between an in-tree composition and a series composition step, see for example node u_2 in Fig. 9.

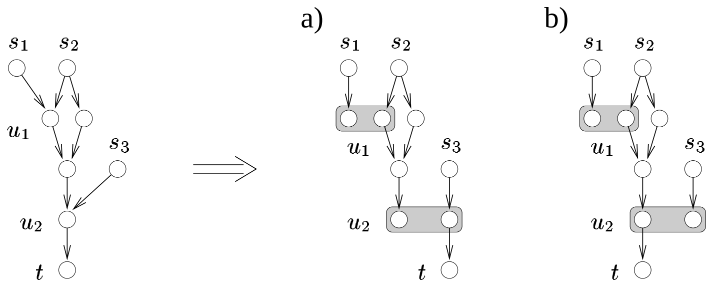


Figure 9: Two different alternatives for decomposing an MSSP-graph

To distinguish between an in-tree and a series composition we define the following set of nodes for a source s:

$$\begin{aligned}
\mathtt{PAR}(s) &:= \{v \mid \mathsf{ELUDE}(v) \cap \mathrm{succ}^*(s) \neq \emptyset\}\ , \\
\mathtt{SER}(s) &:= \{v \mid \mathsf{ELUDE}(v) = \emptyset\ \wedge\ v \in \mathrm{succ}^*(s)\}\ , \\
\mathtt{IN}(s) &:= \{v \mid \exists\, s' \neq s\ [v \in \mathtt{PAR}(s')\ \wedge\ \mathrm{succ}^*(v) = \mathrm{succ}^*(s) \cap \mathrm{succ}^*(s')]\}\ \cup \\
&\ \{v \in \mathtt{SER}(s) \mid \exists\, s' \neq s\ [s' \text{ is a source}\ \wedge\ \mathrm{pred}(v,1) \in \mathrm{succ}^*(s') \setminus \mathrm{succ}^*(s)]\}\ .
\end{aligned}$$

Intuitively speaking, a node v is in $\mathtt{IN}(s)$ if it is a sink of a component in an in-tree composition such that s is a source of this component. Moreover if for v there exist different alternatives for an in-tree composition then the definition of $\mathtt{IN}(s)$ resolves the ambiguity in such a way that v is source of an in-tree component with a sink s if the first predecessor of v does not belong to $\mathrm{succ}^*(s)$; otherwise v is a node of a series composition with a component including s. For example, if in Fig. 9 $\mathrm{pred}(u_2, 1) = s_3$ then we decompose the graph as illustrated in (a), i.e. it is an in-tree composition of a an ordinary SP-graph with source s_3 and sink t and an MSSP-graph with sources s_1, s_2 and sink u_2. Finally we define:

$$\begin{aligned}
\mathrm{sink}(s) &:= \text{first node on the leftmost down path starting at } s\text{, which belong to } \mathtt{IN}(s), \\
&\ \text{resp. the sink of } G\text{, if } \mathtt{IN}(s) \cap \text{lm-down}(s) = \emptyset, \\
\text{in-rank}(s) &:= |\text{lm-down}(s) \cap \mathtt{IN}(s)|\ .
\end{aligned}$$

To generate a decomposition tree for an MSSP-graph G with sources $s_1, s_2, \ldots, s_k$ we compute an ordered sequence $s_{\pi(1)}, s_{\pi(2)}, \ldots, s_{\pi(k)}$ such that $\text{in-rank}(s_{\pi(i)}) \leq \text{in-rank}(s_{\pi(i+1)})$ for all $i \in \{1, \ldots, k-1\}$. Using the implicitly given input ordering we can compute the i'th element of this sequence in logarithmic space for any i. Let $u_i := \mathrm{sink}(s_{\pi(i)})$, $T_i = (V_i, E_i)$ be the decomposition tree of $G_{s_{\pi(i)},u_i}$, and r_i be the root of T_i. The decomposition tree $T = (V_T, E_T)$ can be generated as follows:

$$\begin{aligned} V_T &:= \bigcup_{i \in \{1,\dots,k\}} V_i \quad \cup \quad \{\, x_1, \dots, x_{k-1} \,\} \\ E_T &:= \bigcup_{i \in \{1,\dots,k\}} E_i \quad \cup \quad \{\, (x_i, x_{i+1}) \mid i \in \{1, \dots, k-2\} \,\} \\ & \qquad\qquad\qquad \cup \quad \{\, (r_1, x_1) \,\} \quad \cup \quad \{\, (r_{i+1}, x_i) \mid i \in \{1, \dots, k-1\} \,\} \end{aligned}$$

where x_i are new nodes with $\sigma(x_i) = (\mathtt{i}, u_i)$.

Theorem 13 *A decomposition tree of a MSSP-graph can be computed in $\mathcal{FL}$.*

Proof: That the subgraph induced by the node set $\{x_1, \dots, x_{k-1}, r_1, \dots, r_k\}$ gives a correct decomposition of G into two-terminal SP-graphs can be shown by induction on the length of the chain $x_1, \dots, x_{k-1}$. Assume that $T' = (V'_T, E'_T)$ with

$$\begin{aligned} V'_T &:= \bigcup_{i \in \{1,\dots,k-1\}} V_i \quad \cup \quad \{\, x_1, \dots, x_{k-2} \,\} \\ E'_T &:= \bigcup_{i \in \{1,\dots,k-1\}} E_i \quad \cup \quad \{\, (x_i, x_{i+1}) \mid i \in \{1, \dots, k-3\} \,\} \\ & \qquad\qquad\qquad \cup \quad \{\, (r_1, x_1) \,\} \quad \cup \quad \{\, (r_{i+1}, x_i) \mid i \in \{1, \dots, k-2\} \,\} \end{aligned}$$

is a correct decomposition tree of G' where G' is the MSSP-graph G minus its subgraph with sink u_{k-1} and source $s_{\pi(k)}$. Since, in-rank$(s_{\pi(k)})$ is maximal for all sources s_i there exists no node in $G_{s_{\pi(k)},u_{k-1}}$ that has to serve as an in-tree composition node of G in $G_{s_{\pi(k)},u_{k-1}}$. Note, that according to the defi nition of $\mathtt{IN}(s)$, each series composition in the top level can not be an in-tree composition.

Hence, G can be generated by an in-tree composition of G' and $G_{s_{\pi(k)},u_{k-1}}$ at u_{k-1} – as described by the decomposition tree T. ∎

The counting algorithm for SP-graphs can be extended to this class as well. To compute $\#\mathtt{PATH}$ we use the algorithm for SP-graphs for every subgraph given by a source-sink pair s, t. Note that each subgraph of the form G_{ab} is a two-terminal SP-graph. Therefore we can compute the decomposition tree of each G_{st} in Logspace. Furthermore we can compute the number of paths in G_{st} modulo a given prime number $p \leq |G|$ and also the sum

$$\left(\sum_{\text{source } s \text{ sink } t} \#\mathtt{PATH}(G_{st}) \bmod p \right) \bmod p$$

in logarithmic space. Hence, we can conclude from the algorithm discussed for two-terminal SP-graphs:

Theorem 14 *For MSSP-graphs the function $\#\mathtt{PATH}$ can be computed in $\mathcal{FL}$. The same holds for the size of subgraphs G_{ab} with arbitrary nodes a, b.*

8 Vertex-Series-Parallel Graphs

Another alternative to generalize the family of SP-graphs are vertex-series-parallel graphs:

Definition 7 *A graph $G = (V, E)$ is a* **minimal vertex-series-parallel graph** *(***MVSP** *for short) if it can be constructed using a sequence of the following operations, where $G_i = (V_i, E_i)$ for $i = 1, 2$ are disjoint MVSP-graphs:*

Initial Step: *if G consists of a single node it is an MVSP-graph;*

Parallel Composition: $G = (V_1 \cup V_2, E_1 \cup E_2)$*;*

Series Composition: *let* $\mathbf{sinks}(\boldsymbol{G_i})$ *denote the set of sinks of G_i and* $\mathbf{sources}(\boldsymbol{G_i})$ *its sources, then* $G = (V_1 \cup V_2,\ E_1 \cup E_2 \cup \mathrm{sinks}(G_1) \times \mathrm{sources}(G_2))$.

MVSP-graphs are a subclass of a graph family called CBC-graphs [17].

Definition 8 *A graph G is a* **complete bipartite composite graph (CBC)** *if there exists a set $\{B_1, B_2, \ldots, B_k\}$ of complete bipartite subgraphs of G, called the* **bipartite components** *of G, such that*

1. *every edge of G belongs to exactly one bipartite component,*
2. *for each non-sink node v, all edges leaving v belong to some bipartite component $B_H(v)$, and*
3. *for each non-source node v, all edges entering v belong to some bipartite component $B_T(v)$.*

Lemma 11 *The membership problem for CBC-graphs is in $\mathcal{L}$.*

Proof: To test whether G is a CBC-graph it suffices to test for every node v that for all $v_1, v_2 \in \mathrm{succ}(v) :\ \mathrm{pred}(v_1) = \mathrm{pred}(v_2)$, and that for all $v_1, v_2 \in \mathrm{pred}(v) :\ \mathrm{succ}(v_1) = \mathrm{succ}(v_2)$, which can easily be done in Logspace. ∎

Recognition of MVSP-graphs is based on the concept of a line graph.

Definition 9 *A* **line graph** *$L(G) = (V_L, E_L)$ of a graph $G = (V, E)$ consists of $|E|$ nodes $V_L := \{\ v_e \mid e \in E\ \}$ and the edge set E_L defined by*

$$E_L \ := \ \{(v_{e_1}, v_{e_2}) \mid \exists u, z, w \in V \ \text{ such that } \ e_1 = (u, z) \ \text{ and } e_2 = (z, w)\} \,.$$

Valdes, Tarjan and Lawler have shown in [17] that a graph G is an SP-graph iff its line graph $L(G)$ is an MVSP-graph. Furthermore, for every MVSP-graph G' one can find an SP-graph G'' such that $G' = L(G'')$. Then $G'' = L^{-1}(G')$ will be called a (line graph) **inverse** of G'. $L^{-1}(G')$ is unique when restricted to SP-graphs.

In the following we will concentrate on MVSP graphs and their inverses. To construct the inverse of an MVSP-graph G, consider the bipartite components of G. Each bipartite component B_i is replaced by a node u_i and two such nodes (u_i, u_j) are connected if B_i and B_j are consecutive components, i.e. either

$$\text{sinks}(B_i) \subseteq \text{sources}(B_j) \qquad \text{or} \qquad \text{sources}(B_j) \subseteq \text{sinks}(B_i)\ .$$

The number of edges between two nodes u_i and u_j is given by $|\text{sinks}(B_i) \cap \text{sources}(B_j)|$. Finally we add a new source s and a new sink t and connect s to all other source nodes obtained so far, and connect all sinks to t.

How can this be done in Logspace? For each bipartite component B_i let v_i be the left-most (according to the input ordering) source of B_i. The number of edges (u_i, u_j) between two nodes u_i and u_j of the line graph can be determined by $|\text{succ}(v_i) \cap \text{pred}(\text{succ}(v_j, 1))|$. The source s of the line graph G' is connected to a node u_i by $|\{\ v \in \text{pred}(\text{succ}(v_i, 1)) \mid \text{pred}(v) = \emptyset\ \}|$ many edges. This number is 0 for all bipartite components which are not sources. Finally, there are $|\{\ v \in \text{succ}(v_i) \mid \text{succ}(v) = \emptyset\ \}|$ many edges from a node u_i to t in G'. This transformation can be performed in logarithmic space.

Lemma 12 *For CBC-graphs their inverses can be computed in $\mathcal{FL}$.*

Now to test whether a given graph G is an MVSP we fi rst check whether it is an CBC-graph. If yes then we construct an inverse and test whether this inverse is an SP-graph. Since Logspace is closed under composition we can conclude from section 3:

Theorem 15 *The membership problem for MVSP-graphs is in $\mathcal{L}$.*

To decide whether a node x is reachable from a node y in an MVSP-graph G the following algorithm can be used:

```
procedure MVSP-PATH($y, x$)
1 if $x = y$ then return TRUE
2 else if $y$ is a sink of $G$ or $x$ and $y$ are sinks of the same bipartite component
    then return FALSE
3 else
4    let $G'$ be the inverse of $G$
5    let $x'$ be the node of $G'$ for the bipartite component $B_i$ of $G$ with $x \in$ sink($B_i$)
6    let $y'$ be the node of $G'$ for the bipartite component $B_i$ of $G$ with $y \in$ source($B_i$)
7    if PATH($y', x'$) then return TRUE
8    else return FALSE
```

Since all subroutines can be implemented in Logspace we get

Theorem 16 *Reachability for MVSP-graphs can be decided in $\mathcal{L}$.*

Proof: It remains to show, that MVSP-PATH(y, x) decides the reachability problem correctly. Obviously, x is reachable from y if $x = y$ and unreachable if y is a sink of G different from x, or if both are different sinks of the same bipartite component. On the other hand, the sequence of nodes in a path in the inverse of G represents a sequence of edges in G. Hence, if y is an element of a set of sources of a bipartite component, to which x does not belong, then each path from y' to x' represents a path from the bipartite component of y to x. It follows that if x' is reachable from y' in G' then x is reachable from y in G. ∎

Now consider the problem to compute the decomposition tree of an MVSP-graph.

Definition 10 *A binary tree $T = (V_T, E_T)$ with a labelling function $\sigma : V_T \to \{\mathtt{p}, \mathtt{s}\} \cup V$ is called a* **decomposition tree** *of an MVSP-graph $G = (V, E)$ iff*

1. *for every leaf of T it holds $\sigma(v) \in V$ and for every other node $\sigma(v) \in \{\mathtt{p}, \mathtt{s}\}$,*
2. *G can be generated recursively using T as follows:*
   (a) *if $V_T = \emptyset$ then G is empty;*
   (b) *if T is a single node v then G consists of the single node $\sigma(v)$;*
   (c) *let T_1 be the left and T_2 be the right subtree of T the roots of which are the sons of the root v of T. Furthermore, let G_i be a graph with decomposition tree T_i, then*
      i. *G is the parallel composition of G_1 and G_2 if $\sigma(v) = \mathtt{p}$;*
      ii. *G is the series composition of G_1 and G_2 if $\sigma(v) = \mathtt{s}$.*

In [17] it has been shown that the decomposition tree of an SP-multigraph can by transformed into a decomposition tree of the corresponding MVSP-graph by changing the labelling of the leaves. Recall that each edge of the inverse graph corresponds to a node of the original graph. Furthermore, using the transformation described above we can label each edge in the SP-graph G' by its corresponding node in the MVSP-graph G. Hence, given a decomposition tree of G' we can compute the decomposition tree of G by exchanging the labels at the leaves by the corresponding edge labels of G'.

Theorem 17 *The decomposition tree of an MVSP-graph can be computed in $\mathcal{FL}$.*

Finally, let us consider the problem to compute the number of paths between two nodes of an MVSP-graph. Again we use the inverse graph.

Lemma 13 *Given a MVSP-graph G and its inverse G', then the number of paths between a pair of nodes u, v in G equals the number of paths between the pair u' and v' in G' where u' is the node that corresponds to the bipartite component of G containing u as a source and v' to the bipartite component containing v as a sink.*

Proof: We perform induction on the maximal distance between nodes u and v. If the distance between u and v is 1, i.e. u is a source of the bipartite component with sink v then the number of paths from u to v is one. The strategy used in the Lemma above maps u and v to the same node of G'. Hence, the number of paths from u' to v' is 1 as well.

Let us now assume that claim holds for distance ℓ from u. Then the number of paths from u to a node v with maximal distance $\ell + 1$ from u is the sum of the number of paths from u to the predecessors of v which have maximal distance ℓ. These numbers can be computed by using Lemma 13. It holds:

$$\begin{aligned} \#\mathtt{PATH}_G(u,v) &= \sum_{v_i \in \mathrm{pred}(v)} \#\mathtt{PATH}_G(u,v_i) \\ &= \sum_{\text{bipartite components } B_i} |\mathrm{sinks}(B_i) \cap \mathrm{pred}(v)| \cdot \#\mathtt{PATH}_G(u,u_i)\ , \end{aligned}$$

where u_i is the leftmost sink of B_i according to the input ordering of G. The number $|\mathrm{sinks}(B_i) \cap \mathrm{pred}(v)|$ determines the number of edges between the node v'_i in G' representing B_i and the node v' representing the bipartite component with sink v. It follows that

$$\begin{aligned} \#\mathtt{PATH}_G(u,v) &= \sum_{\text{bipartite components } B_i} |\mathrm{sinks}(B_i) \cap \mathrm{pred}(v)| \cdot \#\mathtt{PATH}_{G'}(u',v'_i) \\ &= \sum_{v'_i \in \mathrm{pred}(v')} |\mathrm{sinks}(B_i) \cap \mathrm{pred}(v)| \cdot \#\mathtt{PATH}_{G'}(u',v'_i) \quad = \quad \#\mathtt{PATH}_{G'}(u',v')\ . \end{aligned}$$

∎

Hence we can conclude

Theorem 18 *For MVSP-graphs the function $\#\mathtt{PATH}$ can be computed in $\mathcal{FL}$.*

Similarly, using MVSP-PATH we can also compute the size of subgraphs of an MVSP graph:

Theorem 19 *For MVSP-graphs the size of subgraphs G_{ab} can be computed in $\mathcal{FL}$.*

9 Conclusions and Open Problems

A deterministic Turing machine working in space $S \geq \log$ can be simulated by an EREW PRAM in time $O(S)$ using at most an exponential number of processors with respect to S (see e.g. [15]). Our space effi cient algorithms for the various graph problems considered here imply logarithmic parallel time algorithms with a polynomial number of processors. The simulation of space-bounded Turing machines by PRAMs can even be performed by the EROW model (exclusive-read owner-write). Hence we can deduce

Corollary 2 *For series-parallel graphs and their extensions considered above, recognition, reachability, decomposition, and path counting can be done in logarithmic time on EROW PRAMs with a polynomial number of processors.*

The exact number of processors depends on the time complexity of the Turing machine. Since our basic log-space algorithms require time $O(n^c)$ for some constant c significantly larger than 1, we probably will not achieve a linear number of processors this way. The reachability problem in series-parallel graphs has been shown to be solvable by an EREW PRAM in logarithmic time using $n/\log n$ processors [16]. But it is still open whether also recognition and decomposition can be done in logarithmic time with a linear number of processors.

If we switch to *undirected* graphs the problems considered here seem to be inherently more difficult. In the undirected case, series-parallel graphs can be characterized as the set of graphs containing no clique of size 4 as a minor [10]. To illustrate the difference between a directed and an undirected series-parallel graph consider the example given in Fig. 10. The first graph G_1 shows a series-parallel directed graph, whereas G_2 is not series-parallel. The third graph G_3 illustrate the undirected graph corresponding to G_2. Note that G_3 is series-parallel, while G_4 is a 4-clique and thus not series-parallel.

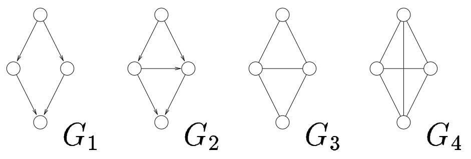


Figure 10: comparing directed and undirected series-parallel graphs

In contrast, for arbitrary graphs the reachability problem seems to be easier in the undirected case than in the directed case. From [1] we know that the undirected version can be solved by a randomized log-space bounded machine, whereas no randomized algorithm is known for the directed case. Are there other distinctions of this kind?

Acknowledgment: Thanks are due to Eric Allender and Markus Bläser for helpful comments and pointers to the literature. Furthermore we would thank Faith Fich, Pierre McKenzie, and Christian Schindelhauer for fruitful discussions.

References

[1] R. Aleliunas, R. Karp, R. Lipton, L. Lovasz, C. Rackoff, *Random Walks, Universal Sequences and the Complexity of Maze Problems,* Proc. 20. FOCS, 1979, 218-223.

[2] C. Álvarez, B. Jenner, *A Very Hard Log-space Counting Classes,* TCS 107, 1993, 3-30.

[3] E. Allender, M. Mahajan, *The Complexity of Planarity Testing,* Proc. 17. STACS, LNCS 1770, 2000, 87-98.

[4] M. Ben-Or, R. Cleve *Computing Algebraic Formulas Using a Constant Number of Registers,* SIAM J. Comput. 21, 1992, 54-58.

[5] H. Bodlaender, B. de Fluiter, *Parallel Algorithms for Series Parallel Graphs,* Proc. 4. ESA, LNCS 1136, 1996, 277-289.

[6] A. Brandst¨adt, V. Bang Le, J. Spinrad, *Graph Classes: A Survey,* SIAM 1999.

[7] S. Buss, S. Cook, A. Gupta, V. Ramachandran, *An Optimal Parallel Algorithm for Formula Evaluation,* SIAM J. Comput. 21, 1992, 755-780.

[8] A. Chiu, G. Davida, B. Litow, *Division in logspace-uniform* NC^1, Theoretical Informatics and Applications 35, 2001, 259-275.

[9] S. Cook, P. McKenzie, *Problems Complete for Deterministic Logarithmic Space,* J. Algo. 8, 1987, 385-394.

[10] R. Duffi n, *Topology of Series-Parallel Networks*, J. Math. Analysis Appl. 10, 1965, 303-318.

[11] D. Eppstein, *Parallel Recognition of Series-Parallel Graphs*, Inf. & Comp. 98, 1992, 41-55.

[12] W. Hesse, *Division Is in Uniform* TC^0, Proc. 28. ICALP, Springer LNCS 2076, 2001, 104-114.

[13] X. He, Y. Yesha, *Parallel Recognition and Decomposition of Two Terminal Series Parallel Graphs*, Inf. & Comp. 75, 1987, 15-38.

[14] B. Jenner, K.-J. Lange, P. McKenzie, *Tree Isomorphism and Some Other Complete Problems for Deterministic Logspace*, publication #1059, DIRO, Université de Montréal, 1997.

[15] R. Karp, V. Ramachandran, *Parallel Algorithms for Shared-Memory Machines*, in: J. van Leeuwen (Ed.): Handbook of Theoretical Computer Science, Volume A, 1990, 869-941.

[16] R. Tamassia and J. S. Vitter, *Parallel Transitive Closure and Point Location in Planar Structures*, SIAM J. Comput. 20, 1991, 708-725.

[17] J. Valdes, R. Tarjan, E. Lawlers *The Recognition of Series Parallel Digraphs,* SIAM J. Comput. 11, 1982, 298-313.