BRICS RS-97-25 Mix Barrington et al.: Searching Constant Width Mazes Captures the $AC^0$ Hierarchy

# Searching Constant Width Mazes Captures the $AC^0$ Hierarchy

David A. Mix Barrington
Chi-Jen Lu
Peter Bro Miltersen
Sven Skyum

BRICS Report Series RS-97-25
ISSN 0909-0878 September 1997

**BRICS**
**Department of Computer Science**
**University of Aarhus**
**Ny Munkegade, building 540**
**DK–8000 Aarhus C**
**Denmark**

**Telephone: +45 8942 3360**
**Telefax: +45 8942 3255**
**Internet: BRICS@brics.dk**

# Searching constant width mazes captures the $AC^0$ hierarchy

David A. Mix Barrington* Chi-Jen Lu*
Peter Bro Miltersen† Sven Skyum†

September 30, 1997

**Abstract**

We show that searching a width $k$ maze is complete for $\Pi_k$, i.e., for the $k$'th level of the $AC^0$ hierarchy. Equivalently, st-connectivity for width $k$ grid graphs is complete for $\Pi_k$. As an application, we show that there is a data structure solving dynamic st-connectivity for constant width grid graphs with time bound $O(\log\log n)$ per operation on a random access machine. The dynamic algorithm is derived from the parallel one in an indirect way using algebraic tools.

# 1 Introduction

Blum and Kozen [4] considered the problem of searching a *maze*. A maze is an object as depicted in Figure 1(a).

Formally, we will define a maze of width $m$ and length $n$ as follows: Let $S_{n,m} = \{1, \ldots, n\} \times \{1, \ldots, m\}$. We call an element $s$ of $S_{n,m}$ a *square* and identify $s$ with the unit square with center $s$ in the plane. A maze of width

*Computer Science Department, University of Massachusetts. Email: {`barring,cjlu`}`@cs.umass.edu`

†BRICS, Basic Research in Computer Science, Centre of the Danish National Research Foundation, Department of Computer Science, University of Aarhus. Email: {`bromille,sskyum`}`@brics.dk`. Supported by the ESPRIT Long Term Research Programme of the EU under project number 20244 (ALCOM-IT).

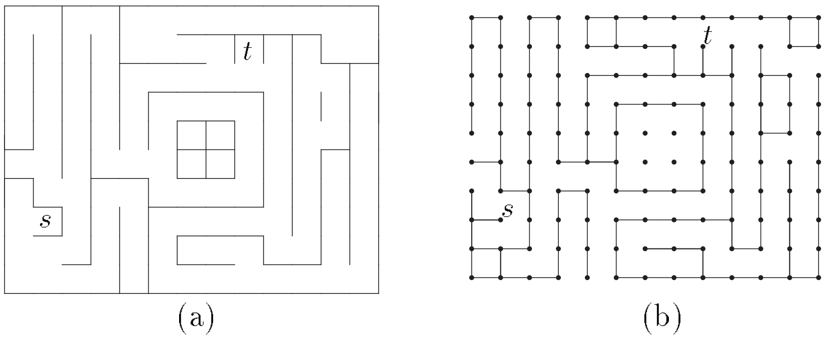


Figure 1: (a) A maze and (b) the corresponding grid graph

$m$ and length $n$ is a set $M$ of line segments (*walls*) of length exactly 1, each separating two squares of $S_{n,m}$. Figure 1(a) depicts a maze of width 10 and length 13 (we consider the longer line segments as consisting of several atomic walls of length 1). A path in the maze between two squares $s$ and $t$ is a path inside the rectangle $[0, n] \times [0, m]$ connecting the centers of $s$ and $t$ and not intersecting any of the walls in $M$. The reader is invited to verify that there is a path between $s$ and $t$ in Figure 1(a). Blum and Kozen gave bounds on the power of systems of automata capable of searching a maze, i.e. capable of deciding whether a path between two given squares in the maze exists. In complexity theoretic terms, one of their main results was that searching a maze is in deterministic logspace.

In this paper we consider the complexity of searching a *constant width* maze, i.e., rather than letting both $n$ and $m$ be parameters, we fix $m$ to a constant $k \geq 1$. Let $\mathrm{MAZE}_k$ be the problem which takes as input (a Boolean encoding of) a maze of width $k$, two squares $s$ and $t$, and decides if there is a path from $s$ to $t$.

We relate the complexity of $\mathrm{MAZE}_k$ in a strong way to the levels of the $AC^0$ hierarchy. Recall the following definitions: Non-uniform $AC^0$ is the class of languages recognizable by families of AND/OR/NOT-circuits of constant depth, polynomial size, and unbounded fan-in. Inside $AC^0$ we find the following hierarchy: Non-uniform $\Sigma_k$ is the class of languages recognizable by circuits with $k$ alternating levels of unbounded fan-in AND and OR gates, with the output an OR-gate and a "zeroth level" of input gates and their negations. Non-uniform $\Pi_k$ is defined analogously, but with the output gate being an AND-gate. Following [1], we define a uniform version of the hiearchy as follows: Uniform $\Pi_k$ ($\Sigma_k$) is the class of languages accepted by alternating Turing machines running in logarithmic time and making exactly $k$ alternations, the first being universal (existential).

An appropriate class of reductions to use for the non-uniform classes in the $AC^0$ hierarchy is the class of (non-uniform) $p$-projections [15]; all the non-uniform classes mentioned above are closed under those. Similarly, an appropriate class of reductions to use for the uniform classes is the class of DLOGTIME-uniform projections (for a precise definition, see Section 5), and all the uniform versions of the classes in the hierarchy are closed under those.

Our main result is:

**Theorem 1** MAZE$_k$ *is complete for non-uniform* $\Pi_k$ *with respect to non-uniform p-projections. Also,* MAZE$_k$ *is complete for uniform* $\Pi_k$ *with respect to DLOGTIME-uniform projections.*

As far as we know, this is the first example of natural complete problems for the levels of the $AC^0$ hierarchy.

There is a close correspondence between mazes and *grid graphs*, as defined by Itai *et al.* [11]. An $n \times k$ grid graph is an undirected graph $G$ with vertex set $V_{n,k} = \{1, \ldots, n\} \times \{1, \ldots, k\}$ and with the property that if $\{(a, b), (c, d)\}$ is an edge in $G$, we have $|a - c| + |b - d| = 1$. The length of the grid graph is $n$ and the width is $k$. A grid graph is shown in Figure 1(b).

The st-connectivity problem USTCON$_k$ for width $k$ grid graphs is the following: Given a grid graph, and two vertices $s$ and $t$, decide if $s$ and $t$ are connected in $G$. There is a trivial isomorphism between MAZE$_k$ and USTCON$_k$: To get from a maze problem to a grid graph problem, simply make a vertex for each square of the maze, and put an edge between two vertices if and only if there is *not* a wall between the corresponding squares. Though the maze formulation is somewhat more appealing, we prefer the grid graph formulation for technical reasons and shall use it in the main part of the paper.

A third setting for these problems is the following variant of bounded-width branching programs. An $n \times k$ *switching network* is a *undirected* labelled graph whose vertices form a rectangular array with $k$ rows and $n$ columns and whose edges are restricted to be between vertices in adjacent columns. (Switching networks are also called “contact schemes” — see the survey of

Razborov [13] for further background.) Each edge is labelled by an input variable, its negation, or the value 1, and the network accepts a given input string iff there is a path from a fixed vertex $s$ to another fixed vertex $t$ such that the label of each edge on the path evaluates to 1 on the input. It is not hard to show that the grid graph problem is closely related to *planar* switching networks as follows: $\mathrm{USTCON}_k$ is complete, under $p$-projections, for the class of languages decidable by families of width-$k$, polynomial-size planar switching networks. This is because an $n \times k$ planar switching network can be simulated by a $kn \times k$ grid graph, and an $n \times k$ grid graph can be simulated by a $kn \times k$ planar switching network. We omit the details of these simulations in this version of the paper.

In our second result, we consider the following *dynamic* graph problem: Maintain, on a random access machine with word size $O(\log n)$, a data structure representing an $n \times k$ grid graph under insertions and deletions of edges and connectivity queries, i.e. queries asking whether there is a path between two vertices, given as input. The equivalent maze problem is the problem of maintaining an $n \times k$ maze under risings of walls, destructions of walls, and path queries, i.e. queries asking whether there is a path between two squares of the maze. For *non-constant* width $m \leq n$, Eppstein *et al* provide a solution to this problem with a time bound $O(\log n)$ per operation [6]. We show:

**Theorem 2** *For any constant $k$, there is a solution to the dynamic connectivity problem for width $k$ grid graphs with time complexity $O(\log \log n)$ per operation. On the other hand, no solution to the dynamic connectivity problem for width 2 grid graphs has time complexity $o(\log \log n / \log \log \log n)$.*

We derive the dynamic algorithm from the parallel one in an indirect, and rather unusual way: We note that the *existence* of the parallel algorithm implies that a certain monoid, $G_k$, associated with the width-$k$ problem is aperiodic by results of Barrington and Therien [2]. Combining this with a result of Thomas [16], we in fact show that $G_k$ has dot-depth exactly $k$, providing a (rather) natural example of such a monoid. Such examples are not encountered too often, so this may be of independent interest. We then use results on dynamic word problems by Frandsen, Miltersen and Skyum [7] to derive the dynamic algorithm. Similar algebraic and language-theoretic tools gives us the lower bound as a corollary to work of Beame and Fich [3].

Though we determine the time complexity of the dynamic problem within a factor of $O(\log\log\log n)$, there is an annoying flaw in the result: The constant in the big-$O$ of the upper bound is $2^{2^{O(k)}}$, while the lower bound is independent of $k$. We leave the existence of a better constant as an open problem.

## 2 Encoding

Since we are dealing with very low level complexity, we have to be a bit careful about the encoding. A grid graph is represented by a number of Boolean *edge indicator variables*, one for each edge position in the grid. The variable is true if and only if the edge is present. The source and sink inputs $s$ and $t$ are given in positional notation; that is, for each vertex $v$ there is an indicator variable which is true if and only if $v = s$ and an indicator variable which is true if and only if $v = t$. Of course, for non-uniform complexity, there is no reason to impose any particular order of these variables. For uniform complexity, we have to specify how the input variables are ordered. Let the edge indicator variables be packed in two binary relations, $E_h \subseteq \{1,\ldots,n-1\}\times\{1,\ldots,k\}$ representing the horizontal edges; $E_h(i,j)$ is true if and only if there is an edge between $(i,j)$ and $(i+1,j)$, and $E_v \subseteq \{1,\ldots,n\}\times\{1,\ldots,k-1\}$; $E_v(i,j)$ is true if and only if there is an edge between $(i,j)$ and $(i,j+1)$. Let the source and sink indicator variables be packed in $S,T \subseteq \{1,\ldots,n\}\times\{1,\ldots,k\}$ in the obvious way. Now we represent an input as the Boolean string consisting of $E_h$, written row by row, concatenated with $E_v$, written row by row, concatenated with $S$, written row by row, concatenated with $T$, written row by row.

## 3 Membership

In this section we show that the connectivity predicate for width $k$ grid graphs is in non-uniform $\Pi_k$. The uniform version of the lemma is deferred to Section 5.

**Lemma 3** USTCON$_k$ *is* $\Pi_k$, *for* $k \geq 3$. *The constructed circuit is positive (monotone) in the edge variables.*

**Proof** We first show that for all $k \geq 1$, the statement "there is a path from vertex $s$ to vertex $t$ in $G$" for *fixed boundary* vertices $s$ and $t$ can be computed by a positive $\Pi_k$ circuit; that is, we do not let $s$ and $t$ be part of the input and we assume them to be on the boundary of the grid. Then, we generalize, first to the case of non-boundary vertices, and then to $s$ and $t$ being given as input.

We show the statement for fixed boundary vertices by induction in $k$, for all $k \geq 1$. Thus, in contrast to the statement of the lemma, we can include $k = 1$ and $k = 2$ as well. This is possible because we don't have to decode information about $s$ and $t$.

Base, $k = 1$: There is a path between $s$ and $t$ if and only if, for all $a$, if $a$ is an edge position between $s$ and $t$, $a$ is an edge. This is a $\Pi_1$ statement in the edge indicator variables, as desired.

Now suppose $k > 1$. Given a grid graph $G$ on $V_{n,k}$, we define its *dual* $G^*$ as follows: $G^*$ has a vertex $s^*$ for each square $s$ of the grid and a vertex $\infty$ representing the region outside the grid. We put an edge between two vertices $u^*$ and $v^*$ of $G^*$ if and only if the edge position separating $u$ and $v$ in $G$ is *not* an edge. Thus, for every edge position $e$ of $G$ there is an edge position $e^*$ of $G^*$ and exactly one of $G$ or $G^*$ has an edge at that position. $G$ and $G^*$ can be simultaneously embedded in the plane. Note that $G^* - \{\infty\}$ is a grid graph on $V_{n-1,k-1}$.

Now, for any given vertices $s$ and $t$ of $G$, there is a path between $s$ and $t$ in $G$ if and only if there is *not* a simple cycle in $G^*$ so that if the cycle is drawn in the plane, $s$ is on the outside of the cycle and $t$ is on the inside of the cycle. Since $s$ and $t$ are border vertices, such a cycle must go through the vertex $\infty$. Let $C$ be some cycle going through $\infty$ and let $e_1^*$ and $e_2^*$ be the two edges adjacent to $\infty$ on the cycle. $C$ separates $s$ and $t$ if and only the edge positions $e_1$ and $e_2$ separate $s$ and $t$ in the following sense: If one tracks the border clockwise from $s$ back to itself, one of $e_1$ and $e_2$ is found before hitting $t$ and the other is found after.

Thus, there is a path from $s$ to $t$ if and only if for all border edge positions $e_1$ and $e_2$, such that $e_1$ and $e_2$ separates $s$ and $t$, there is *not* a path in $G^* - \{\infty\}$ from $u^*$ to $v^*$, where $u$ is the square of $G$ adjacent to $e_1$ and $v$ is the square of $G$ adjacent to $e_2$.

The statement "$e_1$ and $e_2$ separate $s$ and $t$" is independent of the input. Since $G^* - \{\infty\}$ is a grid graph of width $k - 1$ there is a positive $\Pi_{k-1}$ circuit deciding whether a path between two fixed border vertices exists. Note that the inputs of this circuit are edge indicators for $G^*$, i.e. negations of edge indicators for $G$. Thus, using DeMorgan's law, checking whether *no* path between two fixed border vertices exists can be done by a positive $\Sigma_k$ circuit in the primal edge indicator variables. We conclude that the validity of the entire statement can be checked by a positive $\Pi_k$ circuit, as desired.

Now consider the more general problem, where $s$ and $t$ are not on the boundary, but still fixed. Assume without loss of generality that $s$ is to the left of $t$ or right above $t$. Split the graph into 3 parts — the part left of $s$, the part between $s$ and $t$, and the part right of $t$. Compute the transitive closure for each component, restricted to the vertical border vertices. By the above, this can be done by $O(k^2)$ $\Pi_k$ circuits, i.e. a constant number. The end result is now a monotone Boolean function of the computed information. Since the amount of information is constant, we can compute this function with a positive $NC^0$ circuit. Since $\Pi_k$ is closed under positive finite Boolean combinations, the entire thing is $\Pi_k$.

Finally, consider the USTCON$_k$ problem with $s$ and $t$ being part of the input. Recall that they are given by two indicator variables for each vertex. For each value of $s$ and $t$ we can construct a gate $E_{s,t}$ which evaluates to 1 if and only if the inputs are $s$ and $t$; this gate is just an AND of two indicator variables. For each possible value of $(s,t)$, construct the $\Pi_k$ circuit $C_{s,t}$ solving the problem for this value. Now, we adjust $C_{s,t}$ so that it outputs 1, if $s$ or $t$ do not match the actual input. We do this by giving each of the OR gates of the second layer from the top of $C_{s,t}$ one additional input, namely the negation of $E_{s,t}$. The end result is the AND of all these adjusted $C_{s,t}$ circuits. There is no penalty in depth if $k \geq 3$. Note that the final circuit is no longer positive, but the only negative literals are these $E_{s,t}$'s. $\Box$

# 4 Hardness

In this section we show that USTCON$_k$ is hard for non-uniform $\Pi_k$ by non-uniform $p$-projections. The uniform version of the lemma is deferred to Section 5.

**Lemma 4** *For every $k \geq 1$, every problem in non-uniform $\Pi_k$ reduces to* USTCON$_k$ *by a non-uniform $p$-projection.*

**Proof**

We will show the following stronger statement: For every $k \geq 1$, every problem in non-uniform $\Pi_k$ reduces to $\mathrm{USTCON}_k$ and every problem in non-uniform $\Sigma_k$ reduces to $\mathrm{USTCON}_{k+1}$ by $p$-projections. Furthermore, the value of the node $s$ in the reduction is the bottommost left corner of the grid and the value of the node $t$ in the reduction is the bottommost right corner of the grid.

Given a $\Pi_k$ circuit of size $s$, we can construct a $\Pi_k$ formula of size $s^{O(1)}$ computing the same function, so we can assume without loss of generality that we are given a function which can be computed by a $\Pi_k$ formula. By the definition of $p$-projection, an alternative formulation of the statement is then this:

Given a $\Pi_k$ formula $C$, we can construct a polynomial sized, width $k$ grid graph $G(C)$ where some of the edges are labelled with input variables or their negations, the bottommost left corner of the grid is labelled $s$ and the bottommost right corner of the grid is labelled $t$, so that, given an input vector $\mathbf{x}$, if we remove the edges labelled with variables assigned 0, there is a path form $s$ to $t$ in $G(C)$ if and only if $C(\mathbf{x})$ evaluates to true. Similarly, given a $\Sigma_k$ circuit, we can construct a width $k+1$ grid graph with corresponding properties. We will construct this mapping $G$ by recursion in $k$.

First suppose a $\Pi_1$ formula $C$ is given. We can write $C$ as $\bigwedge_{i=1}^{r} x_{j_i} \wedge \bigwedge_{i=1}^{s} \bar{x}_{k_i}$, where $x_{j_i}, i = 1 \ldots r$ and $x_{j_i}, i = 1 \ldots s$ are input variables. The corresponding width 1 grid graph $G(C)$ is shown in figure 2.

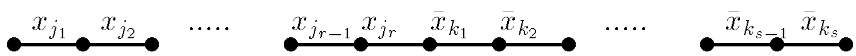

Figure 2: $G(\bigwedge_{i=1}^{r} x_{j_i} \wedge \bigwedge_{i=1}^{s} \bar{x}_{k_i})$

Similarly, if a $\Sigma_1$ formula $C$ is given, we write $C$ as $\bigvee_{i=1}^{r} x_{j_i} \vee \bigvee_{i=1}^{s} \bar{x}_{k_i}$ and let $G(C)$ be the width 2 grid graph of Figure 3.

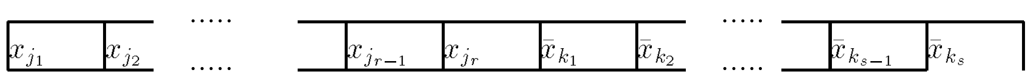

Figure 3: $G(\bigvee_{i=1}^{r} x_{j_i} \vee \bigvee_{i=1}^{s} \bar{x}_{k_i})$

Now, let $k > 1$ and assume we have both the $\Sigma_j$ and $\Pi_j$ constructions for all $j < k$. Let a $\Pi_k$ formula $C$ be given. We can write it as $\bigwedge_{i=1}^r C_i$, where the $C_i$'s are $\Sigma_{k-1}$ formulae. Construct the width $k$ graphs $C(G_i)$ corresponding to the $C_i$'s and let $G(C)$ be the graph of Figure 4. Note that this graph also has width $k$, as desired.


Figure 4: $G(\bigwedge_{i=1}^r C_i)$

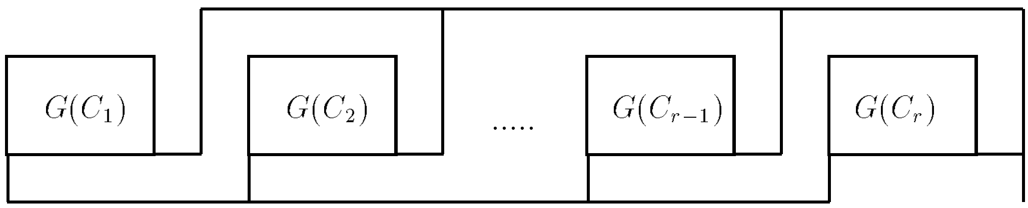


Figure 5: $G(\bigvee_{i=1}^r C_i)$

Finally, let a $\Sigma_k$ formula $C$ be given. Write it as $\bigvee_{i=1}^r C_i$, where the $C_i$'s are $\Pi_{k-1}$ formulae. Construct the width $k-1$ graphs $C(G_i)$ corresponding to the $C_i$'s and let $G(C)$ be the width $k+1$ graph of Figure 5.

The correctness of the construction is easily checked. □

# 5 Uniformity considerations

As in Barrington, Immerman, and Straubing [1], we define a *log-time* Turing machine to have a read-only input tape of length $n$, a constant number of read-write work tapes of total length $O(\log n)$, and a read-write input address tape of length $\log n$. On a given time step the machine has access to the bit of the input tape denoted by the contents of the address tape (or to the fact that there is no such bit, if the address tape holds too large a number). An *alternating* log-time machine has universal and existential states with the usual semantics. Furthermore, the alternating machine queries its input only once in a computation, in its last step.

We now define uniform $\Pi_k$ as the class of languages accepted by alternating log-time Turing machines, making exactly $k$ alternations, the first being universal. It is shown in [1] that this hierarchy is in fact a uniform version of the $AC^0$ circuit hierarchy, where specific questions about the circuit family can be answered by a DLOGTIME Turing machine.

Recall that a family of $p$-projections can be viewed syntactically as a family of maps

$$\sigma_n : \{y_1, y_2, \ldots, y_{m(n)}\} \to \{0, 1, x_1, x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\},$$

where $m(n)$ is bounded by a polynomial in $n$. The syntactic map $\sigma_n$ defines a reduction $\sigma'_n : \{0,1\}^n \to \{0,1\}^{m(n)}$ by

$$\sigma'_n(a_1, a_2, \ldots, a_n)_i = \begin{cases} 0 & \text{if} \quad \sigma_n(y_i) = 0 \\ 1 & \text{if} \quad \sigma_n(y_i) = 1 \\ a_j & \text{if} \quad \sigma_n(y_i) = x_j \\ \bar{a}_j & \text{if} \quad \sigma_n(y_i) = \bar{x}_j \end{cases}$$

A DLOGTIME-uniform projection is a family of projections $\sigma_n$, so that there is a DLOGTIME Turing machine which on input $\langle i, 1^n \rangle$ outputs the binary encoding of the values of $m(n)$ and $\sigma_n(y_i)$ on a specified work tape.

We have the following proposition.

**Proposition 5** *Uniform $\Pi_k$ is closed under DLOGTIME-uniform projections.*

We can now state the uniform version of our main theorem.

**Theorem 6** $\mathrm{USTCON}_k$ *is complete for uniform $\Pi_k$ under DLOGTIME-uniform projections.*

**Proof** (sketch). We must check two things:

1. $\mathrm{USTCON}_k$ can be decided by an alternating log-time Turing machine making exactly $k$ alternations, the first being universal.

2. If a language can be decided by an alternating log-time Turing machine making exactly $k$ alternations, the first being universal, then there is a DLOGTIME-uniform projection $\rho$, so that if $\rho'$ is the reduction: $\{0,1\}^* \to \{0,1\}^*$ defined by $\rho$, $\forall x, x \in L \Leftrightarrow \rho'(x) \in \mathrm{USTCON}_k$

For (1), we verify that the computation done by the circuit can be performed by an alternating log-time machine. The main technical issue is checking whether two border edges, given by their coordinates separate two border vertices, and this can be done by a constant number of integer comparisons, which are easily carried out deterministically by a log-time machine since the integers in question are only $\log n$ bits long. (In fact, we will later use the fact that this "separation" predicate is a Boolean combination of comparisons of column numbers in the graph.) Since only a constant number of such checks must be done during a computation, we are done.

For (2), given an alternating log-time machine, making exactly $k$ alternations, the first being universal, we can modify it by adding a clock, and ensure that all its first $c \log n$ moves are universal, its next $c \log n$ moves are existential, etc. — this blows up the complexity only by a constant factor.

Let $L$ be the language recognized by a such a machine. It is now easy to see that for each $n$, there are DLOGTIME functions $\sigma_n : \{1, \ldots, n^j\}^k \to \{0, 1, \ldots, n\}$ and $\rho_n : \{1, \ldots, n^j\}^k$ so that a string $x = x_1 x_2 \ldots x_n \in \{0,1\}^n$ is in $L$ if and only if

$$\forall_{i_1=1}^{n^c} \exists_{i_2=1}^{n^c} \ldots Q_{i_k=1}^{n^c} x_{\sigma_n(i_1,i_2,\ldots,i_k)}^{\rho_n(i_1,i_2,\ldots,i_k)} \tag{1}$$

Here, for a Boolean value $x$, $x^v$ is $x$ if $v = 0$ and $x$ negated if $v = 1$, and by convention, $x_0 = 0$, this makes it possible to deal with the machine *not* reading a bit at the end of the computation.

Using the technique of Section 4, it is now easy to modify the DLOGTIME machines for $\sigma_n$ and $\rho_n$ into a DLOGTIME-uniform projection reducing $L$ to $\mathrm{USTCON}_{k-1}$. $\square$

# 6 The width-$k$ grid graph monoid

In this section we consider an algebraic interpretation of our results. We explore its consequences in Section 7, but we also consider it interesting in its own right.

We define the width $k$ grid graph monoid $G_k$. It will be a submonoid of the following monoid $M_k$:

The ground set of $M_k$ is the set of equivalence relations on $V_{2,k} = \{1,2\} \times \{1,\ldots,k\}$, or, equivalently, the set of transitively closed undirected graphs with vertex set $V_{2,k}$.

Let $G$ and $H$ be members of $M_k$, viewed as graphs. We now define the composition of $G$ and $H$. Let $U = \{1, \frac{3}{2}, 2\} \times \{1, ..., k\}$. Let $R$ be the graph on $U$ obtained by embedding $G$ in $U$ by the embedding $(1,y) \to (1,y), (2,y) \to (\frac{3}{2}, y)$ and embedding $H$ in $U$ by the embedding $(1,y) \to (\frac{3}{2}, y), (2,y) \to (2,y)$. Let $R^*$ be the transitive closure of $R$. The composition $G \circ H$ is the restriction of $R^*$ to $V_{2,k} = \{1,2\} \times \{1, ..., k\}$.

$G_k$ is now defined to be the submonoid of $M_k$ generated by the transitive closures of the set of grid graphs on $V_{2,k}$.

A very intuitive way of viewing $G_k$ is as follows. An elements of $G_k$ is a collection of plane blobs inside the $[1,2] \times [1,k]$ rectangle in the plane, where a blob is identified with the set of grid points it contains. In Figure 6, (a) are (b) are two such elements. To multiply two elements, we concatenate them and scale down the resulting picture by a factor of two on the x-axis. In Figure 6, (a) and (b) are concatenated to form (c) and then scaled down to (d). Finally, since two blobs are equivalent if they contain the same elements, we can make a nicer picture (e) which is equivalent to (d).

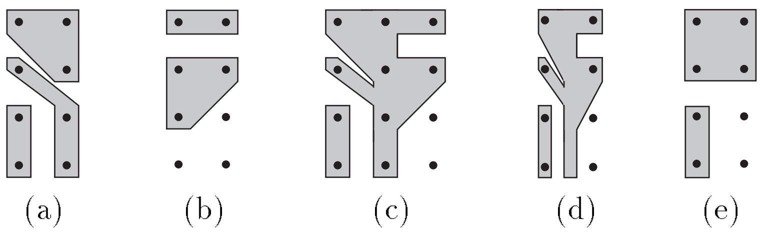


Figure 6: Elements in $G_k$ and their product

Note that every member of $G_k$ can be described as a word $a_1 \circ a_2 \circ \ldots a_{f(k)}$ where the $a_i$'s are closures of $2 \times k$ grid graphs for some function $f$ (a trivial upper bound on $f(k)$ is $|G_k|$). Another way of viewing this: Every member of $G_k$ can be described by the transitive closure of an $f(k) \times k$ grid graph, restricted to the vertical border vertices.

**Lemma 7** *The size of $G_k$ is $c_{2k} = \frac{1}{2k+1}\binom{4k}{2k}$, i.e. the $2k$'th Catalan number.*

**Proof** For this proof we name the vertices $1, 2, \ldots, 2k$ going counter-clockwise and starting in the upper left corner, as in part (a) of Figure 7. Note that the number of ways to connect the $2k$ vertices into blobs does not depend on there being $k$ vertices in each column, but only on the fact that connections among the $2k$ vertices occur only on one side of the line of them (in the planar embedding) and may not intersect each other. We will show that with any number $m$ of vertices, odd or even, the number of distinct arrangements of this kind is $c_m$.

To do this we will biject the graphs with strings in the Dyck language $D_m$, where $D_m$ is the set of words of $m$ pairs of correctly matched parentheses. Consider the highest-numbered vertex to which vertex 1 is connected and call it $j$. (In Figure 7, this is vertex 8.) Form the Dyck language string by concatenating in turn a (, the Dyck string corresponding to the division into blobs of vertices 1 through $j-1$, a ), and the Dyck string for the division into blobs of vertices $j+1$ through $m$. It is easy to verify that this mapping is total and invertible, and hence is a bijection.

It is perhaps easier to carry out this mapping on an example as follows. Relabel the vertices of the graph as in part (b) of Figure 7, so that the label of each vertex in a blob is the smallest original label of any vertex in that blob. Reading the new labels in the original order, we get a word of length $m$ over the set $\{1, \ldots, m\}$ which, as it turns out, characterizes the arrangement. In the example this word is 12332211. To map such a word $w$ to the corresponding Dyck string, first insert $m$ )'s, one after each occurence of a letter in $w$. Then for each letter $a$ occurring in $w$ insert, before the first occurrence of $a$, the string $(^u$, where $u$ is the total number of occurrences of $a$ in $w$. Finally erase all the original letters from the word. In the example, 12332211 eventually becomes ((()((()(())))). It is not hard to verify that the composite mapping from arrangements of blobs to Dyck strings is exactly the mapping defined recursively above. $\Box$

Our earlier analysis of grid graphs can now be interpreted in terms of the algebraic structure of the monoid $G_k$. To use the vocabulary of formal language theory, we may think of a grid graph problem as a string where the individual letters are elements of $G_k$. The following result tells us something about the language of strings representing graphs with a particular connectivity property. It is star-free (meaning that it can be formed from one-letter

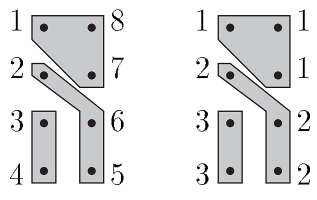


Figure 7: Transforming an element of $G_k$ to a Dyck string

languages by concatenation and boolean operations including complementation), and has dot-depth $k$ (meaning that the optimal depth of nesting of concatenation operations is $k$). See, for example, [12] for further background on algebraic automata theory.

**Lemma 8** *$G_k$ is an aperiodic monoid with dot-depth exactly $k$.*

**Proof** (sketch) We first show that the dot-depth is at most $k$. In our proof of Theorem 6, we showed that connectivity in a width-$k$ grid graph could be expressed by a logical formula with $k$ quantifiers in prenex normal form, and atomic predicates that either referenced individual edges (properties of individual "letters" in the input string) or compared two column numbers (positions of letters in the string). By a theorem of Thomas [16], any language so describable has a syntactic monoid that is aperiodic with dot-depth $k$. But this syntactic monoid is $G_k$ itself, since $G_k$ was designed to exactly capture this connectivity information.

If the dot-depth of $G_k$ were less than $k$, we could derive a contradiction as follows. Consider any circuit $C$ of depth $k$ and size $s$. By our construction in Lemma 4, we can construct a word over $G_k$, of size polynomial in $s$, whose product determines the value of $C$. But if $G_k$ has dot-depth $k-1$ or less, this product can be evaluated by a circuit of depth $k-1$ and size polynomial in $s$, using a construction of Barrington and Thérien [2]. Since $C$ was arbitrary, we have collapsed two distinct levels of the $AC^0$ hierarchy, contradicting a theorem of Sipser [14]. $\Box$

In Section 7, we only use the aperiodicity of $G_k$, not its dot-depth. Still, $G_k$ gives us an example of a natural (or at least easily visualizable) monoid which we know is aperiodic with dot depth exactly $k$ — such examples are rare.

# 7 The dynamic grid graph connectivity problem

We consider the following *dynamic* graph problem: Maintain, on a random access machine with word size $O(\log n)$, a data structure representing an $n \times k$ grid graph under insertions and deletions of edges and connectivity queries, i.e. queries asking whether there is a path between two given vertices.

**Lemma 9** *For any constant $k$, there is a solution to the dynamic connectivity problem for width $k$ grid graphs with time bound $O(\log \log n)$ per operation. The constant in the big-O is $2^{2^{O(k)}}$.*

**Proof** We shall use a result of Frandsen, Miltersen, and Skyum [7]. First some terminology. Let $S$ be a finite monoid. Let $S$-RANGE be the problem of maintaining, on a random access machine with word size $O(\log n)$, a sequence $(a_1, a_2, \ldots, a_n) \in S^n$ under a change$(i, b)$-operation which changes $a_i$ to $b \in S$, and a range query operation query$(i, j)$ which returns $a_i \circ a_{i+1} \circ \cdots \circ a_{j-1} \circ a_j$. Frandsen, Miltersen, and Skyum show:

**Fact 10** *If $S$ is aperiodic, there is a solution to $S$-RANGE with time bound $O(\log \log n)$ per operation. The constant in the big-O is $2^{O(|S|)}$.*

We show that dynamic reachability reduces to dynamic range queries over $G_k$ with a constant overhead. Since $G_k$ is aperiodic, we are done. Suppose we are to maintain a graph on $\{1, .., n\} \times \{1, ..., k\}$. We maintain the product $a_1 \circ a_2 \circ ... \circ a_{n-1}$ where $a_i$ is the element of $G_k$ correponding to the subgraph in $\{i, i+1\} \times \{1, ..., k\}$. A change in the graph corresponds to a single change of an $a_i$. If we are to answer if there is a path from $(x_1, y_1)$ to $(x_2, y_2)$ we do the following (assume wlog that $x_1 < x_2$): We query the subproducts $a = a_1 \circ ... \circ a_{x_1-1}, b = a_{x_1} \circ ... \circ a_{x_2}$ and $c = a_{x_2+1} \circ ... \circ a_n$. Whether or not there is a path between $(x_1, y_1)$ and $(x_2, y_2)$ is completely determined by $(a, b, c, y_1, y_2)$, and since these are all in a constant range, we can hardwire the answer for each possible value of the tuple into the algorithm. $\Box$

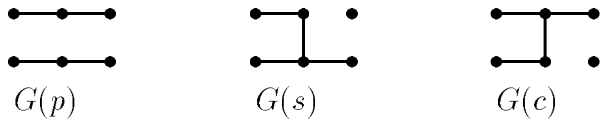


Figure 8: Gadgets for reducing the dynamic prefix problem for $L$ to dynamic connectivity

**Lemma 11** *Assume $k \geq 2$. There is no solution to the dynamic connectivity problem for width $k$ grid graphs with time bound $o(\log n \log n / \log\log\log n)$ on a RAM with word size $O(\log n)$.*

**Proof** We shall use a result by Beame and Fich [3]. First some terminology. Call a regular language $L \subseteq \Sigma^*$ *indecisive* if and only if for all $x \in \Sigma^*$, there exist $z$ and $z'$ such that $xz \in L$ and $xz' \notin L$. Given a language $L$, the dynamic $(L, n)$-prefix problem is the problem of maintaining a string $x = x_1 \ldots x_n \in \Sigma^n$ under a change$(i, a)$ operation which changes $x_i$ to $a$ and a prefix$(j)$ operation which answers the question "Is $x_1 x_2 \ldots x_j \in L$?". Beame and Fich show:

**Fact 12** *If $L$ is indecisive, then, in any implementation of the dynamic $(L, n)$-prefix problem on a RAM with word size $O(\log n)$, if the change operations each take time at most $2^{(\log n)^{1-\Omega(1)}}$, then the query operation takes time at least $\Omega(\log\log n / \log\log\log n)$.*

Now let $L$ be the regular language $(c + s + p)^* sp^* + p^*$, i.e. the language over $\{c, s, p\}$, where $x \in L$ if and only if the last letter of $x$ which is not a $p$ is an $s$. This language is indecisive, so Beame and Fich's result apply. Now assume that the dynamic connectivity problem for width 2 grid graphs can be solved with time bound $o(\log n \log n / \log\log\log n)$ per operation. We will show that the dynamic $(L, n)$-prefix problem can also be solved with time bound $o(\log n \log n / \log\log\log n)$, a contradiction.

Given an instance $a_1 a_2 \cdots a_n$ of $(L, n)$-prefix to maintain, we maintain a grid graph $G$, of width 2 and length $2n$, defined as follows. Consider $G$ to be divided into $n$ blocks of length 2. The $i$'th block of $G$ is $G(a_i)$, where $G$ is the mapping defined in Figure 8. Clearly, a change of a symbol in the maintained string corresponds to a constant number of insert and delete operations in the dynamic graph. Now, in order to determine if $a_1a_2\dots a_i \in L$, we remove all edges in the $i+1$'st block of $G$ using the delete operation of the dynamic connectivity operation. Then we ask if there is a path from the bottom left vertex of $G$ to the bottom right vertex of the $i$'th block of $G$. This is the case if and only if $a_1a_2\dots a_i \in L$. After getting the right answer, we restore the data structure by reinserting the edges of block $i$. This completes the reduction. □

# 8 Generalization to directed graphs

In a directed grid graph, each edge present has one or both of the two possible orientations, and we consider finding directed paths from $s$ to $t$. (Such graphs correspond to planar nondeterministic branching programs or planar "switching-and-rectifier networks" [13], except that a directed grid graph need not have horizontal arrows in only one direction. It follows from the analysis here, of course, that this additional ability is of no use in the case of constant width.)

All of our theorems about constant-width grid graphs hold for directed grid graphs. Of course, the lower bounds are trivial extensions, but we must revisit the upper bounds:

**Lemma 13** STCON$_k$ *is in uniform* $\Pi_k$*, with the constructed circuit being positive in the edge variables.*

**Proof** (sketch) Given a directed grid graph $G$, we will define its dual $G^*$ as follows. The possible edge positions of $G^*$ are exactly as in the undirected case. Now let $e^*$ be an edge position of $G^*$, we define which orientations of $e^*$ is present in $G^*$ as a function of the orientations of $e$ present in $G$: An orientation of $e^*$ is present if and only if the orientation, turned $90°$ degrees clockwise, is *not* an orientation of $e$.

Now it is easy to see that there is a directed path from $s$ to $t$ if and only if there is *not* a directed cycle in $G^*$, going clockwise around $t$, with $s$ on the outside. The rest of the proof proceeds exactly as in the proof of Lemma 1 — the only operations needed on the column numbers are comparisons. □

As before, we can define a monoid whose elements are now directed $2 \times k$ grid graphs, and show that this monoid is aperiodic with dot-depth exactly $k$.

As a corollary, we also get the same upper bound on the directed dynamic grid graph connectivity as on undirected dynamic grid graph connectivity. (Interestingly, for general graphs, the directed version of the dynamic problem seems to be much harder than the undirected version).

# 9 Discussion and open problems

Interesting open problems include:

- As mentioned in the introduction, Blum and Kozen showed that the general problem, where the width is not fixed, is in $L$. By carrying out our construction in Section 4 for a width of $O(\log n)$, we get that even this restricted version of the general problem is hard for $NC^1$. An obvious open question is to determine the complexity of the general problem precisely: Is it complete for $NC^1$, or for $L$, or does it have intermediate complexity?

- We can also consider the general version of the problem for directed grid graphs, which is still hard for $NC^1$ but which we only know to be in $NL$. Our notion of duality gives a simple positive reduction of this problem to its complement, suggesting (but of course not proving, as $NL$ is in fact closed under complement, by a more complicated reduction) that it is not $NL$-complete. There are potentially interesting restrictions of this problem as well, where we prohibit edges in one or two of the four directions.

- A similar gap occurs in our understanding of the dynamic grid graph connectivity problem, if the width of the graph is non-constant. If the width is a free parameter $m$, with the restriction $2 \leq m \leq n$, the following is known: Eppstein *et al* [6] construct a data structure with a time bound of $O(\log n)$ per operation and Eppstein [5] shows a lower bound of $\Omega(\log m / \log \log m)$. This lower bound is improved by Husfeldt and Rauhe [8] to $\Omega(m)$, provided $m \leq \log n / \log \log n$. As we pointed out, our upper bound is $2^{2^{O(m)}} \log \log n$. This improves the general bound only for $m \ll \log \log \log n$. From Beame and Fich [3] follows a lower bound of $\Omega(\log \log n / \log \log \log n)$, provided $m \geq 2$. Combining everything, we get an upper bound of $O(\min(\log n, 2^{2^{O(m)}} \log \log n))$ and a lower bound of $\Omega(\min(m, \log n/\log\log n) + \log\log n/\log\log\log n)$ with a big gap to close.

- $G_k$ is a, rather natural, aperiodic monoid of dot-depth exactly $k$, and the word problem for any aperiodic monoid of dot-depth $k$ reduces to the word problem for $G_k$. Is there a purely algebraic way of viewing this curious fact?

## 10 Acknowledgements

We would like to thank Paul Beame and Arny Rosenberg for help with historical references, Howard Straubing and Denis Therien for very helpful discussions about automata theory and monoids, and Sairam Subramanian for very helpful discussions about dynamic graph problems. This work was greatly facilitated by the March 1997 Dagstuhl workshop in Boolean Function Complexity, attended by the first and third authors.

## References

[1] D. A. M. Barrington, N. Immerman and H. Straubing. On uniformity within $NC^1$ *Journal of Computer and System Sciences*, 41(3):274–306.

[2] D. A. M. Mix Barrington and D. Thérien. Finite monoids and the fine structure of $\mathrm{NC}^1$. *Journal of the ACM*, 35(4):941–952, October 1988.

[3] P. Beame and F. Fich. On searching sorted lists: A near-optimal lower bound. Manuscript, 1997.

[4] M. Blum and D. Kozen. On the power of the compass (or why mazes are easier to search than graphs). In *19th Annual Symposium on the Foundations of Computer Science*, pages 132–142, October 1978.

[5] D. Eppstein. Dynamic connectivity in digital images. Technical Report 96-13, Univ. of California, Irvine, Department of Information and Computer Science, 1996.

[6] D. Eppstein, G. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic planar graph. *Journal of Algorithms*, 13:33–54, 1992.

[7] G. S. Frandsen, P. B. Miltersen, and S. Skyum. Dynamic word problems. *Journal of the ACM* 44:257–271, 1997.

[8] T. Husfeldt and T. Rauhe. Hardness of dynamic computation. Manuscript, 1997.

[9] N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, 1987.

[10] N. Immerman and S. Landau. The complexity of iterated multiplication. *Information and Computation*, 116(1):103–116, January 1995.

[11] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[12] J. E. Pin. *Varieties of Formal Languages.* New York: Plenum Press, 1986.

[13] A. A. Razborov. Lower Bounds for deterministic and nondeterministic branching programs. In L. Budach, ed., *Fundamentals of Computation Theory, 8th International Conference: FCT '91.* Lecture Notes in Computer Science 529, 47–60. Berlin, Springer Verlag, 1991.

[14] M. Sipser. Borel sets and circuit complexity. In *Proceedings, 15th ACM Symposium on the Theory of Computing*, 1983, 61–69.

[15] S. Skyum and L. G. Valiant. A complexity theory based on Boolean algebra. *Journal of the ACM*, 32(2):484–502, April 1985.

[16] W. Thomas. Classifying regular events in symbolic logic. *J. Comput. System Sci.* 25, 1982, 360–376.

## Recent BRICS Report Series Publications

**RS-97-25** **David A. Mix Barrington, Chi-Jen Lu, Peter Bro Miltersen, and Sven Skyum.** ***Searching Constant Width Mazes Captures the $AC^0$ Hierarchy*****. September 1997. 20 pp.**

**RS-97-24** **Søren B. Lassen.** ***Relational Reasoning about Contexts*****. September 1997. 45 pp. To appear as a chapter in the book *Higher Order Operational Techniques in Semantics*, eds. Andrew D. Gordon and Andrew M. Pitts, Cambridge University Press.**

**RS-97-23** **Ulrich Kohlenbach.** ***On the Arithmetical Content of Restricted Forms of Comprehension, Choice and General Uniform Boundedness*****. August 1997. 35 pp.**

**RS-97-22** **Carsten Butz.** ***Syntax and Semantics of the logic $\mathcal{L}^{\lambda}_{\omega\omega}$*****. July 1997. 14 pp.**

**RS-97-21** **Steve Awodey and Carsten Butz.** ***Topological Completeness for Higher-Order Logic*****. July 1997. 19 pp.**

**RS-97-20** **Carsten Butz and Peter T. Johnstone.** ***Classifying Toposes for First Order Theories*****. July 1997. 34 pp.**

**RS-97-19** **Andrew D. Gordon, Paul D. Hankin, and Søren B. Lassen.** ***Compilation and Equivalence of Imperative Objects*****. July 1997. iv+64 pp. Appears also as Technical Report 429, University of Cambridge Computer Laboratory, June 1997. To appear in *Foundations of Software Technology and Theoretical Computer Science: 17th Conference*, FCT&TCS '97 Proceedings, LNCS, 1997.**

**RS-97-18** **Robert Pollack.** ***How to Believe a Machine-Checked Proof*****. July 1997. 18 pp. To appear as a chapter in the book *Twenty Five Years of Constructive Type Theory*, eds. Smith and Sambin, Oxford University Press.**

**RS-97-17** **Peter Bro Miltersen.** ***Error Correcting Codes, Perfect Hashing Circuits, and Deterministic Dynamic Dictionaries*****. June 1997. 10 pp.**

**RS-97-16** **Noga Alon, Martin Dietzfelbinger, Peter Bro Miltersen, Erez Petrank, and Gábor Tardos.** ***Linear Hashing*****. June 1997. 22 pp. A preliminary version appeared with the title *Is Linear Hashing Good?* in *The Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 465–474.**