Dynamic Word Problems*

Gudmund Skovbjerg Frandsen
Peter Bro Miltersen
Sven Skyum

Computer Science Department, Aarhus University

May 1993

Abstract

Let M be a fixed finite monoid. We consider the problem of implementing a data type containing a vector $x = (x_1, x_2, \ldots, x_n) \in M^n$, initially $(1, 1, \ldots, 1)$ with two kinds of operations, for each $i \in \{1, \ldots, n\}, a \in M$, an operation $\texttt{change}_{i,a}$ which changes x_i, to a and a single operation `product` returning $\prod_{i=1}^{n} x_i$. This is the dynamic *word* problem. If we in addition for each $j \in 1, \ldots, n$ have an operation $\texttt{prefix}_j$; returning $\prod_{i=1}^{n} x_i$, we talk about the dynamic *prefix* problem. We analyze the complexity of these problems in the *cell probe* or *decision assignment tree* model for two natural cell sizes, 1 bit and log n bits. We obtain a classification of the complexity based on algebraic properties of M.

*This work was partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 7141 (project ALCOM II) and CCI-Europe.

1 Introduction and results

Statement of problem

A *finite monoid* is a finite set M equipped with an associative binary operation $\circ$ and containing an identity element 1. We assume M is non-trivial, i.e. $M \neq \{1\}$. The *word problem* for M is to take a sequence of elements $(x_1, x_2, \ldots, x_n)$ and find the product $x_1 \circ x_2 \circ \cdots \circ x_n$ (we will write this as $x_1, x_2, \cdots, x_n$ when no confusion can arise). The more general *prefix problem* for M is to take a sequence of elements $(x_1, x_2, \ldots, x_n)$ and find the sequence $(x_1, x_1x_2, x_1x_2x_3, \ldots, x_1x_2 \cdots x_n)$.

In this paper, we consider the *dynamic* complexity of word and prefix problems. The dynamic *word* problem for a monoid M is the task of implementing a data type M-WORD(n) containing a vector $x = (x_1, x_2, \ldots, x_n) \in M^n$, initially $(1, 1, \ldots, 1)$ with two kinds of operations:

- For each $i \in \{1, \ldots, n\}$ and $a \in M$ an operation $\texttt{change}_{i,a}$. This operation changes x_i to a.
- A single operation `product`, returning $x_1x_2 \cdots x_n$.

The dynamic *prefix* problem for M is the task of implementing a data type M-PREFIX(n) defined in the same manner, except that we instead of the single product operation have an operation $\texttt{prefix}_j$ for each $j \in \{1, \ldots, n\}$ returning $x_1x_2 \cdots x_j$.

The model in which we consider implementing the data type is the *cell probe* or *decision assignment tree* model, previously considered by Fredman [9, 10] and Fredman and Saks [11]. In this model, the complexity of a computation is the number of cells accessed in the random access memory containing the data structure during the computation, while the computation itself is for free (and information about which operation to perform is also given for free). The number of bits B in a cell is a parameter of the model. Formally, the model is as follows: In an implementation of the data type we assign to each operation a decision assignment tree, i.e. a rooted tree containing *read* nodes and *write* nodes. When performing an operation we proceed from the root of its tree to one of the leaves. The read nodes are labeled with a location of the random access memory. Each has 2^B sons, one

for each possible content of the memory location. The write nodes which are unary are labeled with a memory location and a value between 0 and $2^B - 1$. When such a node is encountered, the value is written in the memory location. If the operation is to return an answer, this is found in the leaf finally encountered. The complexity of an implementation is the depth of its deepest tree, i.e. we consider only worst case, deterministic complexity. The model is very general, focusing on the *storage and access* aspect of data structuring. We consider two natural choices of B, namely $B = 1$ and $B = \log n$. Lower bounds in the $B = \log n$ model is also lower bounds for the complexity of implementing the type on a *random access computer* [1], i.e. a unit cost random access machine with word size bounded by $O(\log n)$. Also, our upper bounds in the $B =$log n models do not take any advantage of the non-uniformity of the model, i.e. the data structures can be implemented on a random access computer.

Motivation

There are two main motivations for studying dynamic word problems. Firstly, dynamic word problems can be regarded as a special case of a very general class of natural data structure problems, dynamic language membership problems considered by Sairam, Vitter and Tamassia [18] and Miltersen [15, 16]. A problem in this class is given by a language $L \subseteq \{0,1\}^*$. We are supposed to implement a data type L-MEMBER containing a string $x \in \{0,1\}^*$ with three kinds of operations:

- `init`(n). This operation initializes x to 0^n.
- `change`(i,a). This operation changes the i'th component of x to a.
- `query`. This operation returns `true` if $x \in L$, `false` otherwise.

Many natural occurring d ata structure problems for instance dynamic graph problems, can be phrased as a dynamic language membership problem. Sairam, Vitter and Tamassia consider the complexity class *incr-POLYLOG-TIME* which is the class of languages having an implementation on a log cost random access machine so that `init`(n) can be done in time $n^{O(1)}$ and `change` and `query` can be done in time $\log^{O(1)} n$ (with $n = |x|$). Clearly,

incr-POLYLOGTIME $\subseteq P$, but it is an open problem if the inverse containment holds. This problem corresponds roughly to the P versus NC question in parallel complexity. We seem to be far from answering the P versus *incr-POLYLOGTIME* question. Indeed, the best known lower bound for a language in P for the number of cells touched by a `change` or `query` operation seems to be $\Omega(\log n)$, even when the cell size is 1 [15, 16]. This again corresponds to the world of parallel complexity where no lower bound on (bounded fan-in) depth for problems in P better than $\Omega(\log n)$ is known, i.e. $P = NC^1$ is unresolved. However, in order to get a better understanding of the techniques available and their limitations, one has studied sub-NC^1 complexity extensively and have a very good understanding of it. It seems natural to try the same approach for dynamic problems, i.e. study the languages for which the dynamic membership problem can be implemented so that the `change` and `query` operations touches $O(\log n)$ cells or less. With the natural Boolean encoding, the dynamic word problems is a nice, natural class of such problems.

The second motivation is closely related to the first: Several authors have noted that there seems to be some kind of correspondence between parallel and dynamic complexity. Is this correspondence purely accidental, or can we establish some kind of formal correspondence, linking parallel and dynamic complexity? Indeed, Cohen and Tamassia [7] and Sairam, Vitter and Tamassia [18] consider this question and has several partial results on when one can and when one can not transfer information between the two worlds. In order to gain further understanding, dynamic word and prefix problems are nice, because the *parallel* complexity of these problems has been the subject of extensive research during the last decade. From a parallel point of view, there is not much difference between word and prefix problems, since the prefixes can be computed independently. A tight relationship between the complexity and algebraic properties of the monoid has been shown in a series of paper. Chandra, Fortune and Lipton [6] note that if the monoid contains a non trivial group as subset, the word problem is not in AC^0, i.e. can not be computed by polynomial size, unbounded fan-in, constant depth, AND-OR circuits. In fact, depth $\Theta(\log n/ \log \log n)$ is sufficient and necessary if polynomial size is to be retained. (this is as a corollary to a celebrated series of results by Furst, Saxe and Sipser [12], Yao [19] and Håstad [13]). Chandra, Fortune and Lipton show that if the monoid is in fact *group free*, i.e. does not contain a non trivial group, the prefix problem is in AC^0 and

the size of the circuit can be made almost linear. Barrington and Therien [4] refine this result by relating the *exact* depth required by a polynomial size, unbounded fan-in circuit solving the word problem of a group free monoid to the position of the monoid in a hierarchy, the "dot-depth" hierarchy. For monoids containing groups, Barrington [2] shows that for a solvable monoid, i.e. a monoid where all contained groups are solvable, the word problem is in ACC, i.e. can be computed by polynomial size, constant depth circuits containing AND, OR and MOD-q gates for some fixed integer q, while the word problem for a non-solvable monoid is complete for NC^1. Thus, modulo the unsolved problem "$ACC = NC^1$?", we have a complete understanding of the parallel complexity of the word problem for all finite monoids M, based on algebraic properties of M. So, is it the same algebraic properties of the monoid influencing the complexity in the dynamic case? Interestingly, it turns out that the complexity of the dynamic word problem *is* influenced in a nice way by the algebraic properties of the monoid, and in some cases the relevant properties are the same. However, we get a more diverse picture, because, interestingly, the properties influencing the complexity change with cell size and when going from word to prefix problems.

Results

While dynamic word problems do not seem to have been considered previously, the literature contains two previous results on dynamic prefix problems in the cell probe model (by $\mathbf{Z}_r$ we mean the group of integers modulo r).

- If M is any non-trivial finite monoid then for cell size $B = 1$, M-PREFIX(n) can be implemented with complexity $O(\log\ n)$. Furthermore, in any implementation, some operation will require access to $\Omega(\frac{\log\ n}{\log\ \log\ n})$ cells. This result is due to Fredman [10] (who only states it for $M = \mathbf{Z}_2$ but the proof is easily seen to hold in general).
- If M is any finite monoid, then for cell size $B = \log\ n$, M-PREFlX(n) can be implemented with complexity $O(\frac{\log\ n}{\log\ \log\ n})$. If $M = \mathbf{Z}_r$, for some $r \geq 2$, then in any implementation, some operation will require access to $\Omega(\frac{\log\ n}{\log\ \log\ n})$ cells. This result is due to Fredman and Saks [11] (who again only state it for $M = \mathbf{Z}_2$ but the proof holds in general). Since any

[1]We actually show the bound for a slightly larger class of monoids, see Theorem 10

non-trivial group contains $\mathbf{Z}_r$ for some r as a subgroup, the complexity is completely determined for all monoids containing non-trivial groups.

With the results we prove in this paper, we get the bounds in Figure 1. For instance, for any finite monoid M, which is commutative, but not a group, M-WORD(n) can be implemented on a cell probe machine with cell size $B = 1$ with the operations having worst case complexity $2 \log\log n + O(1)$. Furthermore, in any correct implementation, some operation will require access to at least $\frac{1}{2} \log\log n - O(1)$ cells in the worst case. The constants in the O's may depend upon M.

Word problem

Cell size	Type of monoid	Lower bound	Upper bound
$B = 1$	commutative group	$\Theta(1)$	
	commutative non-group	$\frac{1}{2} \log\log n - O(1)$	$2 \log\log n + O(1)$
	non-commutative	$\Omega(\frac{\log n}{\log\log n})$	$O(\log n)$
$B = \log n$	commutative	$\Theta(1)$	
	group free		$O(\log\log n)$
	non-commutativegroup[1]	$\Theta(\frac{\log n}{\log\log n})$	
	other		$O(\frac{\log n}{\log\log n})$

Prefix problem

Cell size	Type of monoid	Lower bound	Upper bound
$B = 1$	any non-trivial	$\Omega(\frac{\log n}{\log\log n})$	$O(\log n)$
$B = \log n$	group free		$O(\log\log n)$
	contains non-trivial group	$\Theta(\frac{\log n}{\log\log n})$	

Figure 1: The classification

The proof techniques are rather diverse: The $B = 1$ upper bound for commutative non-groups is based on an efficient implementation of a *counter*, i.e. a data type containing an integer which can be incremented, decremented and tested for equality to certain special values. The corresponding lower bound is based on a Ramsey-like theorem, the Erdös-Rado sunflower lemma.

The $B = 1$ lower bound for non-commutative monoids is based on a technique due to Fredman [10], used previously for getting the $B = 1$ prefix lower bound mentioned above. The $B = \log n$ upper bound for group free monoids is proved by an application of the Krohn-Rhodes decomposition theorem [14] (which was also used to show that the prefix problem for the same class of monoids is in AC^0 [6]). The data structure use *stratified trees* [17]. The lower bound for non-commutative groups is proved by reducing $\mathbf{Z}_r$-PREFIX(n) to M-WORD(n).

The only significant gap for $B = 1$ is the gap for non-commutative monoids. We do not have any additional information here, i.e. there is no non-commutative monoid for which we have an upper bound better than $O(\log n)$, nor do we for any monoid know a lower bound better than $\Omega(\log n/ \log \log n)$. For $B = \log n$, we understand the prefix problem fairly well. If an $\Omega(\log \log n)$ bound for any non commutative monoid is proven, we would have tight bounds on the prefix problem for any finite monoid M. For the word problem and $B = \log n$, we have good bounds on all monoids except for a certain class of non-commutative monoids containing non-trivial commutative groups but no non-commutative ones.

As an extension to the above theory, we consider the dynamic membership problem for regular languages $L \subseteq \{0,1\}^*$. Clearly, we can get an upper bound for this problem by solving the dynamic word problem for the *syntactic monoid* M_L of L, i.e. the monoid consisting of maps on the states of the minimum finite automaton for L, generated by the two maps corresponding to the 0-transitions and the 1-transitions. However, we have examples that show that this is not necessarily optimal, i.e. a regular language may be easier than its syntactic monoid. A similar situation arises in parallel complexity, e.g. a regular language may be in AC^0, while the word problem for its syntactic monoid is not [6]. However, in the parallel case, the situation has been completely resolved by Barrington, Compton, Straubing and Thérien [3]. We provide a sufficient condition for a regular language to be as hard as its syntactic monoid in the dynamic case, but we don't know anything in general about languages violating the condition. We consider this a promising topic for further research.

2 Proofs

Commutative groups

Proposition 1 *If M is a commutative group, then for any $B \geq 1$, there is an implementation of M-WORD(n) on a B bit machine with complexity O(1).*

Proof The data structure contains two components, the value of the x_i's themselves in an array and the value $w = \prod_{i=1}^{n} x_i$. When x_i is changed from a to b, multiply w by $a^{-1}b$. □

Commutative non-groups

Both the upper and lower bound for commutative non-groups are based on counting. For the upper bound, we use that for any $a \in M$, the sequence $a^0, a^1, \ldots, a^i, \ldots$ is eventually cyclic. Thus we can find out the contribution of a's to the final product by maintaining the value of the number of a's modulo the length of the cycle and looking out for those special values which may occur before we enter the cycle.

Theorem 2 *If M is a commutative monoid, then for cell size $B = 1$, M-WORD(n) can be implemented with complexity $2 \log \log n + O(1)$. For cell size $B = \log n$, there is an implementation with complexity $O(1)$.*

Proof Let $a \in M$. There are values $1 \leq k_a, c_a \leq |M|$ and a map $f_a : \{0, \ldots, c_a - 1\} \to M$ so that $i \geq k_a \Rightarrow a^i = f_a((i - k_a) \bmod c_a)$. The data structure consists of the following components:

n	0	1	2	3	4	5	6	7	8	9	10	11
$w(n)$	00$\underline{0}$	00$\underline{1}$	01$\underline{0}$	01$\underline{1}$	0$\underline{1}$0	1$\underline{0}$0	1$\underline{0}$0	1$\underline{0}$1	11$\underline{0}$	11$\underline{1}$	1$\underline{1}$0	$\underline{1}$00

Figure 2: Scheme for representing integers

- An array containing the x_i's.
- For each $a \in M - \{1\}$ the value of $(|\{i | x_i = a\}| - k_a) \bmod c_a$ represented in any convenient way. These values are easily maintained.

- For each $a \in M - \{1\}$, the value of $n_a = |\{i|x_i = a\}|$ represented in a way to be determined later.

When we change x_i from a to b, we increment n_b and decrement n_a. In order to find the product, for each $a \in M - \{1\}$, we check if the number of a's is between 0 and $k_a - 1$ in which case we find the exact number and know the contribution of a's to the product. Otherwise the contribution is determined by $n_a - k_a$ modulo c_a. If we have cell size $B = \log n$, we can trivially maintain the n_a's in time $O(1)$. For cell size $B = 1$, the maintenance of the n_a's becomes non-trivial. What we need is the implementation of a data type COUNTER(n, V), maintaining a value $t \in \{0, \ldots, n\}$ with V a subset of $\{0, \ldots, n\}$ of size $O(1)$ with the following operations:

- `inc`. This operation puts $t := t + 1$.
- `dec`. This operation puts $t := t - 1$.
- `test`. If $t \in V = \{v_0, \ldots, v_{k-1}\}$ with $t = v_j$, this operation outputs j.

We now show that all COUNTER(n, V) can be implemented on a cell probe machine with cell size $B = 1$ with the `inc` and `dec` operations taking time $\log \log n + O(1)$ and the test operation taking time $O(1)$. This completes the proof of the theorem.

Let m be a positive integer. We consider the following scheme for representing an integer $t \in \{0, \ldots 2^{m+1} - 2 - m\}$ as a bit string $w(t)$ of length m with one of the bits marked:

- $w(0) = 00 \ldots 0\underline{0}$
- If $w(t) = x_{m-1} \ldots x_1\underline{0}$ then $w(t+1) = x_{m-1} \ldots x_1\underline{1}$.
- If $w(t) = x_{m-1} \ldots x_l 0\underline{1} \ldots x_0$ then $w(t+1) = x_{m-1} \ldots x_l 1\underline{0} \ldots x_0$.
- If $w(t) = x_{m-1} \ldots x_l\underline{0}0 \ldots x_0$ then $w(t+1) = x_{m-1} \ldots x_l 0\underline{0} \ldots x_0$.
- If $w(t) = x_{m-1} \ldots x_l 1\underline{1} \ldots x_0$ then $w(t+1) = x_{m-1} \ldots x_l\underline{1}0 \ldots x_0$.

It is relatively easy to see that this scheme defines a unique representation for each integer in the range. The case $m = 3$ is summarized in figure 2.

The idea of the implementation of COUNTER(n, V) is to pick m so that $2^{m+1} - 2 - m \geq n$ and to represent t as a bit array containing $w(t)$ and $\lceil \log m \rceil$ bits representing the position of the mark. The mark is represented in Gray code (using $\log \log n$ bits) so that after having read it, it can be incremented and decremented in time $O(1)$. With this representation, t can be incremented and decremented in time $\log \log n + O(1)$ as required.

We still have to explain how to implement the `test`-operation: For each value $v \in V$ we maintain a bit array $A_v[0..m-1]$ with the following invariant attached (where h is the position of the mark):

- Let i be the greatest i strictly smaller than h such that for $j \leq i, w(v)_j = w(t)_j$. Then $A_v[i] = 1$. Let k be the smallest k strictly greater than h such that for all $j \geq k, w(v)_j = w(t)_j$. Then $A_v[k] = 1$. For $i < j < k, A_v[k] = 0$.

It is easy to see that we can maintain this structure with complexity $O(1)$ during `inc`'s and `dec`'s. If we furthermore maintain a bit array $H[0..m-1]$ with $H[i] = 1$ if and only if $h = i$, we can check if $t = v$ in time $O(1)$. Since $|V| = O(1)$, we can identify t if $t \in V$ in time $O(1)$. □

The counter in the proof is a special case of a more general class of counters, described by Frandsen, Miltersen, Schmidt and Skyum [8].

We next prove that if M is a commutative non-group, the complexity of any implementation of M-WORD(n) is $\Omega(\log \log n)$. The proof of this lower bound is based on the following idea: If M is commutative but not a group, there is an element $a \in M$, so that $a^i = 1$ implies that $i = 0$. Now suppose we have a state where the sequence of elements consists solely of a's and we successively changes a's into 1's. For the first $n-1$ operations, we get an answer different from 1, while after the last operation, we get the answer 1. Thus, we have *almost* reduced the type COUNTER$(n, \{0\})$ with initial state n and no `inc`-operation to the word problem, except for the fact that we don't have a single `dec`-operation, but a family of n operations, and we should use each one only once. It is easy to see that COUNTER$(n, \{0\})$ can not be implemented without the `dec` operation taking time $\log \log n$ in the worst case, since the data structure must consist of at least $\log n$ bits. If we can generalize this lower bound for counters to this very special counter, we are done. The problem is that the different `dec`-operations do not necessarily have anything to do with one another. Indeed, they may address different

segments of the data structure and interact in complicated ways. However, if we by S_i denote the set of memory locations addressed by the decision assignment tree corresponding to the i'th `dec`-operations, we can use a theorem from the combinatorial theory of set systems which ensures that we can find a subfamily of the S_i's which behaves "nicely". More precisely we are going to apply the following definition and lemma:

Definition 3 (Erdös and Rado, Boppana and Sipser) A *sunflower* with p petals is a collection $S_1, S_2, \ldots, S_p$ of (not necessarily distinct) sets so that the intersection $S_i \cap S_j$ is the same for each pair of distinct indices i and j. The intersection is called the center of the sunflower.

Lemma 4 (Erdös and Rado) *Let $S_1, \ldots, S_n$ be a collection of (not necessarily distinct) sets each of cardinality at most l. If $n > (p-1)^{l+1} l!$, then the collection contains as a subcollection a sunflower with p petals.*

For a proof, see Boppana and Sipser [5]. Note that they state a slightly different version of the lemma, because they require the collection to be of distinct sets. However, only the base case of the induction has to be modified to convert their proof into a proof of the above lemma.

Theorem 5 *If M is a commutative monoid, but not a group, then with cell size $B = 1$, in any implementation of M-WORD(n) some operation accesses at least $\frac{1}{2} \log \log n - 1 - o(1)$ cells in the worst case.*

Proof Let an implementation I of M-WORD(n) be given. Assume its worst case complexity is less than or equal to d.

For $i \in \{1, \ldots, n\}$, let S_i be the union of the set of memory locations appearing in the decision assignment tree corresponding to the operation $\texttt{change}_{i,1}$ and the decision assignment tree corresponding to the single product-operation, i.e. the set of memory locations which could possibly appear during the execution of these two operations. We have that $|S_i| \leq l$ where $l = 2^{d+1}$. By the lemma, we find a sunflower $S_{i_1} \ldots S_{i_p}$ with p petals, where $\log p \geq \frac{\log n}{l+1} - O(\log(l+1))$. Let C be the center of the sunflower.

Since M is commutative but not a group, there is an element $a \in M$, so that $\forall b \in M : ab \neq 1$. Now we consider performing the sequence of operations $\texttt{change}_{i_1,a}, \ldots, \texttt{change}_{i_p,a}$ starting in the initial state of the data

structure. Let $s_0 \in \{0,1\}^{|C|}$ be the state of C after these i operations. In this state, we now perform $\texttt{change}_{i_1,1}$ and let s_1 be the new state of C. Next, we perform $\texttt{change}_{i_2,1}$ and let s_2 be the new state of C etc.

We now show that $s_j \neq s_k$ for $0 \leq j < k < p$. Indeed, assume $s_j = s_k$. Then, starting from the initial state of the data structure, the sequence of operations

$$\texttt{change}_{i_1,a}, \ldots, \texttt{change}_{i_p,a}, \texttt{change}_{i_1,1}, \ldots, \texttt{change}_{i_p,1}, \texttt{product}.$$

and the sequence of operations

$$\begin{gathered}\texttt{change}_{i_1,a}, \ldots, \texttt{change}_{i_p,a} \\ \texttt{change}_{i_1,1}, \ldots, \texttt{change}_{i_j,1}, \texttt{change}_{i_{k+1},1}, \ldots, \texttt{change}_{i_p,1}, \texttt{product}\end{gathered}$$

return the same answer. Therefore $1 = a^{k-j}$, a contradiction, since a has no inverse. □

Therefore $|C| \geq \log p$ and hence $l \geq \log p$ and thus $l \geq \frac{\log\, n}{l+1} - O(\log(l+1))$, i.e. $l \geq (1-o(1))\sqrt{\log\, n}$ and $d \geq \frac{1}{2} \log \log\, n - 1 - o(1)$.

There are other natural dynamic problems besides word problems to which counting "almost" reduces in the same way. For examples, see the thesis of Miltersen [16].

Non-commutative monoids, cell size B = 1

For the lower bound, we are going to apply a technique due to Fredman [10]. Fredman's technique is best described as reduction from a special data type, WHICH-SIDE(n), maintaining a value $t \in \{1, \ldots, n\}$ with two operations:

- For each $i \in \{1, \ldots, n\}$, an operation $\texttt{init}_i$, putting $t := i$. The first operation performed on the type *must* be an $\texttt{init}_i$ for some i.
- For each $j \in \{1, \ldots, n\}$, an operation $\texttt{ask}_j$, answering `true` if $j < t$ and `false` otherwise. All operations performed after the first one *must* be of this type.

The proof of Theorem 5 in [10] is easily modified to show the following lemma.

Lemma 6 (Fredman) *In any implementation of* WHICH-SIDE(n) *with cell size $B = 1$, at least $\Omega(\frac{\log\ n}{\log\ \log\ n})$ cells are accessed by some operation in the worst case.*

Theorem 7 *If M is a non-commutative monoid, in any implementation of M-*WORD*(n) at least $\Omega(\frac{\log\ n}{\log\ \log\ n})$ cells is accessed in the worst case.*

Proof Let $ab \neq ba$ be a non-commutative pair from M.

Assume an implementation I of M-WORD(n) is given, we will show how to use it to implement WHICH-SIDE(n). Apart from I we also need a bit array $A[1..n]$, initially 0. The $\texttt{init}_i$-operation is performed by executing $\texttt{change}_{i,a}$, on I and setting $A[i] := 1$. The $\texttt{ask}_j$-operation is performed by testing if $A[j] = 1$, if it is, we return the answer `false`. Otherwise we execute $\texttt{change}_{j,b}$ on I and finds out if the product maintained by I is ab or ba by executing `product`, after which we know the answer. Before returning the answer, we execute $\texttt{change}_{j,1}$. □

Group free monoids, cell size $B = \log\ n$

We wish to implement M-PREFIX(n) for a group free monoid M.

We use the following lemma, a consequence of the Krohn-Rhodes decomposition theorem [14] (recall that a *semigroup* is an associative structure that does not necessarily have an identity element).

Lemma 8 (Krohn-Rhodes) *Let S be a group free finite semigroup. Then one of the following cases hold.*

- $S = \langle a \rangle = \{a, a^2, a^3, \ldots, a^k = a^{k+1}\}$
- *For all $a, b \in S, ab = a$.*
- $S = V \cup T$, *where $V \neq S$, $T \neq S$, V is a subsemigroup of S, i.e. closed under the semigroup operation, while T is a left ideal of S, i.e. $ab \in T$ for each $a \in S,\ b \in T$.*

Furthermore, we use the following auxiliary data type, NEIGHBOUR(n), containing a set $K \subseteq \{1, 2, \ldots, n\}$ and supporting the following operations

- For each $j \in \{1, \ldots, n\}$ an operation $\texttt{insert}_j$ putting $K := K \cup \{j\}$ and an operation $\texttt{delete}_j$ putting $K := K - \{j\}$.
- For each $j \in \{1, \ldots, n\}$ an operation $\texttt{pred}_j$ returning $\max\{i \mid i \leq j, i \in K\}$ and an operation $\texttt{succ}_j$ returning $\min\{i \mid i \geq j, i \in K\}$.

NEIGHBOUR(n) can be implemented on a machine with cell size $O(\log n)$ with all operations being $O(\log \log n)$ using the *stratified tree* data structure of Van Emde Boas, Kaas and Zijlstra [17].

Theorem 9 *If M is a group free monoid, M-*PREFIX*(n) can be implemented on a random access computer with cell size $O(\log n)$, so that all operations take time $O(\log \log n)$.*

Proof We actually implement a more general data type, M-INFIX(n), where we for each pair (i, j) have an operation $\texttt{infix}_{i,j}$ returning $\prod_{k=i}^{j} x_k$.

Given a semigroup S, let S' be the monoid defined by adjoining an identity element 1_S to S. We prove the theorem by proving the following statement:

- For a semigroup S, S'-INFIX(n) can be implemented so that all operations take time $O(\log \log n)$.

The proof is by induction in the size of S. Thus suppose that the theorem holds for all semigroups smaller than S. Now let us construct a data structure for S'-INFIX(n). By the decomposition lemma, there are three possible cases for S.

- $S' = \{1_S, a, a^2, a^3, \ldots, a^k = a^{k+1}\}$. In this case the data structure is the following one: an array $A[1..n]$ containing the x_i's and a NEIGHBOUR(n) structure containing the indices i for which $x_i \neq 1_S$. The `change` operation is $O(\log \log n)$. For the $\texttt{infix}_{i,j}$, we find the first k successors of i (i inclusive) which are different from 1_S. If all these occur before j, the answer is a^k, otherwise we can compute the product. The complexity is $O(\log \log n)$.
- For all $a, b \in S$, $ab = a$. In this case the data structure is again an array $A[1..n]$ containing the x_i's and a NEIGHBOUR(n) containing the

non 1_S-indices. We compute the value of an infix by finding the first non 1_S-entry in the interval and returning this if it exists, returning 1_S otherwise.

- Otherwise $S = V \cup T$. By the induction hypothesis, we have structures for V'-INFIX(n) and T'-INFIX(n) at our disposal. Now define

$$y_i = \begin{cases} x_i & \text{if } x_i \in V - T \\ 1_V & \text{otherwise} \end{cases}$$

$$z_i = \begin{cases} x_i & \text{if } x_i \in T \\ 1_T & \text{otherwise} \end{cases}$$

$$K = \{i \mid x_i \in T \wedge x_{i+1} \notin T\}$$

$$u_i = \begin{cases} 1_T & \text{if } i \notin K \\ \prod_{j=k+1}^{i} x_i & \text{where } k = \max\{r \mid r < i, r \in K \cup \{0\}\} \end{cases}$$

We keep the z_i's and the u_i's in two T'-INFIX(n) structures, the y_i's in a V'-INFIX(n) structure and the set K in a NEIGHBOUR(n) structure. In addition, we also keep the x_i's themselves in an array. It is now possible to show that we can the maintain these structures during changes of the x_i's and answer `infix`-queries on the x_i's by making a constant number of calls to the substructures and a constant amount of overhead. The details will appear in the full paper. □

Non-commutative monoids containing groups

Theorem 10 *If M is a monoid containing elements a,b such that*

- $\langle a \rangle$ *is a group with one-element* $1_a = a^k$.
- $1_a ba \neq ab1_a$

then in any implementation of M-WORD(n) on a machine with cell size $B = \log n$, *some operation will require access to at least* $\Omega(\log n / \log\log n)$ *cells in the worst case.*

Proof By a^{-i} we mean the inverse of a^i in the group $\langle a \rangle$. Thus, $a^{-i}a^i = a^i a^{-i} = 1_a$. With this in mind, for $i \geq 1$, define $a_i = a^i b a^{-i}$. Let $d \geq 1$ be minimal so that $a_1 = a_{d+1}$. Clearly, $d \leq k$. Furthermore, $a_i \neq a_j$ for $1 \leq i < j \leq d$, since otherwise we would have $a_1 = a_{d+1+i-j}$. Finally, $d \geq 2$, since otherwise we would have $aba^{-1} = a^2ba^{-2} \Rightarrow 1_a ba = ab1_a$. We now show how to implement $\mathbf{Z}_d$-PREFIX(n) using an implementation of M-WORD(n). By the Fredman-Saks lower bound for this problem, we are done. We are to maintain $(x_1, x_2, x_3, \ldots, x_n) \in (\mathbf{Z}_d)^n$. We do this by maintaining $(y_1, y_2, y_3, \ldots, y_n) \in M^n$ in an M-WORD(n)-structure while maintaining the invariant $y_i = a^{x_i}$. We change x_i to t by changing y_i to a^t. Suppose now we are to compute $\sum_{i=1}^{j} x_i \bmod d$. Suppose the current value of x_j is t. We find the value $w = \prod_{i=1}^{n} y_i$. We change y_j to $a^t b w^{-1}$ and find the value $w' = \prod_{i=1}^{n} y_i$. This is now $a^r b a^{-r}$ if and only if $\sum_{i=1}^{j} x_i \equiv r \pmod d$. After this we change the value of y_j back. □

Regular languages

In this section, we discuss the connections between dynamic word problems and the dynamic membership problem for regular languages. We will, without loss of generality, restrict our discussion to regular languages over the alphabet $\{0, 1\}$.

Let us first recall the following definition from the theory of formal languages:

$\circ$	1	a	b
1	1	a	b
a	a	a	b
b	b	a	b

Figure 3: The syntactic monoid of $\{0,1\}^*0$

Definition 11 Let $L \subseteq \{0,1\}^*$ be a regular language and let A be the

minimal deterministic automaton for recognizing L with state set Q and transition function $\delta : Q \times \{0,1\} \to Q$.

Given a string $w = w_1 w_2 \dots w_n \in \{0,1\}^n$, w induces a map $\phi_L(w) : Q \to Q$ defined inductively by

- $\phi_L(\epsilon)(q) = q$
- $\phi_L(w_1 \dots w_n)(q) = \delta(\phi_L(w_1 \dots w_n)(q), w_n)$

Note that ϕ_L is a morphism from the free monoid with generators 0 and 1 into the monoid of maps on Q. It is called the *syntactic morphism* of L. The image of ϕ_L, i.e. $\phi_L(\{O,1\}^*)$ is called the *syntactic monoid* of L or M_L.

Clearly, we can solve the dynamic membership problem for L by solving M_L-WORD(n). Thus, upper bounds on the complexity of the latter gives upper bounds on the complexity of the former. For example, as a corollary to Theorem 9, if L is a *star free* regular language, i.e. if L can be described using the symbols $\cdot, \cup, -, \emptyset, \sum^*$, the dynamic membership problem for L can be solved in time $O(\log \log n)$ on a random access machine, since the syntactic monoid of a star free language is group free [3]. However, bounds obtained in this way are not necessarily optimal as the following example tells us:

Let $L = \{0,1\}^*0$. The dynamic membership problem for L can be solved in constant time. However, the syntactic monoid of L is isomorphic to the monoid in Figure 3, which is non-commutative so any $B = 1$ implementation has complexity $\Omega(\frac{\log\ n}{\log\ \log\ n})$.

It is interesting to note that a similar phenomenon arises when considering the *parallel* complexity of regular languages [6], a regular language may be in AC^0 even when the word problem for its syntactic monoid is not. In the parallel case, however, the complexity of regular languages is now well understood [3].

It would be nice to have a similar complete understanding of the dynamic complexity of regular languages recognition. However, we will settle for the following theorem which gives a sufficient condition for a regular language to be as hard as its syntactic monoid. We don't know anything in general about the complexity of languages violating the condition.

Theorem 12 *Let L be a regular language. If there is an integer t, so that*

$\phi_L(\{0,1\}^t) = M_L$, *then for any cell size* $B \geq 1$, *if L-MEMBER can be implemented with* `change` *and* `query` *having complexity f(n), where* $f(O(n)) = O(f(n))$, *then* M_L-WORD*(n) can be implemented with complexity* $O(f(n))$.

Proof Let Q be the state set of the minimal automaton for recognizing L. Let s be the initial state and let F be the set of accepting states. Let U be a finite set of words, so that for all $q \in Q$ there is a $w \in U$ with $\phi_L(w)(s) = q$. Let V be a finite set of words, so that there for all $p, q \in Q$ with $p \neq q$ is a word $w \in V$ with $\phi_L(w)(p) \in F$ but $\phi_L(w)(q) \notin F$ or vice versa (since the automaton is minimal, we can find the sets U and V). For each $y \in M_L$, let $\tau(y)$ be a word in $\{0,1\}^t$ with $\phi_L(\tau(y)) \in y$.

We now construct a data structure for maintaining $y = y_1 \dots y_n$ with $y_i \in M_L$. We use $|U| \cdot |V|$ data structures for the dynamic language membership problem of $L \cap \{0,1\}^m$ for various values of m. More precisely, for each pair $(u, v) \in U \times V$, we maintain the membership in L of the string $u\tau(y_1)\tau(y_2)\dots\tau(y_n)\ v$. We now show that if we know the membership of each of these strings in L we know the value of $y_1 \dots y_n$. Recall that y is a map from Q to Q so if we know $y(q)$ for all q in Q, we know y. Recall also that $y(q) = \phi_L(\tau(y_1)\tau(y_2)\dots\tau(y_n))(q)$. Given q, let u be a string with $\phi_L(u)(s) = q$, i.e. $y(q) = \phi_L(u\ \tau(y_1)\tau(y_2)\dots\tau(y_n))(s)$ Now the following procedure for determining $y(q)$ suggests itself: Initially every $p \in Q$ is a candidate for $y(q)$. We now repeatedly pick a pair p_1, p_2 and eliminate one of them by taking a string v with $\phi_L(v)(p_1) \in F$ but $\phi_L(v)(p_2) \notin F$ and checking if $\phi_L(u\ \tau(y_1)\tau(y_2)\dots\tau(y_n)\ v)(s)$ is in F, i.e. checking if $u\ \tau(y_1)\tau(y_2)\dots\tau(y_n)\ v$ is in L. □

Note that in the example above, the condition in the theorem is violated, since we have $\phi_L(\{0,1\}^t) = \{a, b\} \neq M_L$ for every value of $t \geq 1$.

3 Conclusion and open problems

We have obtained a fairly good classification of the complexity of dynamic word and prefix problems for a monoid M, based on algebraic properties of M. The following problems remain.

- Close the gap between $\Omega(\log\ n/\log\log\ n)$ and $O(\log\ n)$ for the case $B = 1$. Is a further distinction between different monoids necessary or

can we get the same bounds for them all?

- For $B = \log n$, get a $\Omega(\log \log n)$ lower bound on the word (or prefix) problem for non commutative monoids.
- We have not yet classified the $B = \log n$ complexity of the word problem for those non-commutative monoids which contains a non-trivial group, and where for any such cyclic group $\langle a \rangle$ with one element 1_a, $1_a ba = ab1_a$ for any $b \in M$.
- Classify the complexity of the dynamic membership problem for regular languages.
- Are there any formal links to parallel complexity? Some properties seem to be the same, for instance the prefix problem for a monoid M is in AC^0 if and only if the monoid is group free, otherwise, if polynomial size is to be obtained, depth $\Theta(\log n/\log \log n)$ is sufficient and necessary. Similarly, the dynamic prefix problem for cell size $B = \log n$ is $O(\log \log n)$ if and only if the monoid is group free, otherwise, time $\Theta(\log n/\log \log n)$ is sufficient and necessary. Other properties are quite different, for instance, the $B = 1$ dynamic complexity of the word problem for M does not seem to have much to do with its parallel complexity.

Acknowledgements

The second author would like to thank Desh Ranjan and Shiva Chaudhuri for helpful discussions.

References

[1] D. Angluin, L.G. Valiant: Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings, *J. Comput. System Sci.* **18** (1979) 155-193.

[2] D.A. Barrington, Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1, *J. Comput. System Sci.* **38** (1989) 150-164.

[3] D.A. Mix Barrington, K. Compton, H. Straubing, D. Therien, Regular Languages in NC^1, *J. Comput. System Sci.* **44**(1992) 478-499.

[4] D.A. Mix Barrington, D. Therien, Finite monoids and the fine structure of NC^1, *J. Assoc. Comput. Mach.* **35** (1988) 941-952.

[5] R.B. Boppana, M. Sipser: The Complexity of Finite Functions, in: J. van Leuuwen, ed., *Handbook of Theoretical Computer Science, Vol. A* (Elsevier, Amsterdam, 1990) 757-804.

[6] A.K. Chandra, S. Fortune, R. Lipton, Unbounded fan-in circuits and associative functions, in: *Proc. l9th ACM Symp. on Theory of Compting* (1987) 123-131.

[7] R.F. Cohen, R. Tamassia, Dynamic Expression Trees and their Applications, in *Proc. 2nd Annual ACM-SIAM Symp. on Discrete Algorithms* (1991) 52-61.

[8] G.S. Frandsen, P.B. Miltersen, E.M. Schmidt, S. Skyum, Counting on a 1 bit machine, in preparation.

[9] M.L. Fredman: Observations on the complexity of generating quasi-Gray codes, *SIAM J. Comput.* **7** (1978) 134-146.

[10] M.L. Fredman: The Complexity of Maintaining an Array and Computing its Partial Sums, *J. Assoc. Comput. Mach.* **29** (1982) 250-260.

[11] M.L. Fredman, M.E. Saks: The Cell Probe Complexity of Dynamic Data Structures, in: *Proc. 21st Ann. ACM Symp. on Theory of Computing* (1989) 345-354.

[12] M. Furst, J. Saxe, M. Sipser, Parity, circuits and the polynomial time hierarchy, *Math. Systems Theory* **17** (1984) 13-27.

[13] J. Håstad, *Computational limitations for small depth circuits,* (MIT Press, Cambridge, MA, 1986).

[14] K. Krohn, J. Rhodes, Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines, *Trans. Am. Math. Soc.* **116** (1965) 450-464.

[15] P.B. Miltersen, On-line reevaluation of functions, Aarhus University Tech. Report DAIMI PB-380.

[16] P.B. Miltersen, *Combinatorial Complexity Theory,* Aarhus University Ph.D. Thesis, 1993.

[17] P. Van Emde Boas, R. Kaas, E. Zijlstra, Design and implementation of an efficient priority queue, *Math. Systems Theory* **10** (1977) 99-127.

[18] S. Sairam, J.S. Vitter, R. Tamassia, A complexity theoretic approach to incremental computation, in: *Proc. 10th Ann. Symp. Theoretical Aspects of Computer Science* (1993) 640-649.

[19] A.C. Yao, Separating the polynomial-time hierarchy by oracles, in: *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science* (1985) 1-10.