NONDETERMINISTIC CIRCUITS, SPACE COMPLEXITY AND QUASIGROUPS

by

Marty J. Wolf

Computer Sciences Technical Report #870

August 1989

Nondeterministic Circuits, Space Complexity and Quasigroups

Marty J. Wolf[1]

Computer Sciences Department
University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706
August 15, 1989

Abstract: By considering nondeterminism as a quantifiable resource we introduce new nondeterministic complexity classes obtained from NC circuits using a bounded number of nondeterministic gates. Let NNC(f(n)) denote the class of languages computable by an NC circuit family with O(f(n)) nondeterministic gates. If f(n) is limited to log n, we show that the class obtained is equivalent to NC. If f(n) is allowed to encompass all polynomials we show the class obtained is equivalent to NP. The class of most interest, NNC(polylog), obtained by letting f(n) encompass all polylog functions, contains a version of the quasigroup (Latin square) isomorphism problem. The quasigroup isomorphism problem is not known to be in P or NP-complete, thus NNC(polylog) is a candidate for separating NC and NP. We also show that NNC(polylog) $\subseteq$ DSPACE(polylog). In fact, if we denote the complexity class obtained from an NC^k circuit with O($\log^j n$) nondeterministic gates by NNC^k($\log^j n$), we show that NNC^k($\log^k n$) is contained in DSPACE($\log^k n$) which leads to DSPACE($\log^2 n$) algorithms for the quasigroup isomorphism problem, the Latin square isotopism problem and the Latin square graph isomorphism problem. The only other known bound for these problems is Miller's time bound of O($n^{\log_2 n + O(1)}$). This result is also a generalization of Lipton, Snyder and Zalcstein's DSPACE($\log^2 n$) algorithm for the group isomorphism problem. We also show that for every k NNC(n^k) $\subseteq$ DSPACE(n^k), and if for some k there exists a j such that DSPACE(n^k) $\subseteq$ NNC(n^j) then NP = PSPACE.

SECTION 1. Introduction.

By giving NC circuits a limited amount of nondeterminism, we develop potentially interesting complexity classes between NC and NP, as well as demonstrate their relationship to well known DSPACE classes. As a result of the relationship between these new classes and

[1] Research supported in part by NSF Grants No. DCR-8504485 and DCR-8552596

DSPACE classes, we develop an DSPACE($\log^2 n$) algorithm for the quasigroup (Latin square) isomorphism problem. Lipton, Snyder and Zalcstein [11] have a DSPACE($\log^2 n$) algorithm for group isomorphism, but, as Miller [12] points out, their technique relies on the associativity of groups and does not readily generalize to quasigroups. Miller goes on to present a more general technique requiring $O(n^{\log_2 n + O(1)})$ time for the quasigroup isomorphism problem. We use nondeterministic NC circuits to show that quasigroup isomorphism, Latin square isotopism and Latin square graph isomorphism can be decided in DSPACE($\log^2 n$).

Expanding the approach of Kintala and Fischer in [5], [8], and [9], we consider nondeterminism as a quantifiable resource. The usual assumption is that a particular computational device is able to make either as many nondeterministic moves as needed or none at all. Fischer and Kintala [5] studied polynomially time bounded real-time Turing machine computations with a bounded number of nondeterministic moves and developed an infinite hierarchy of languages between the real-time languages (the class of languages accepted by deterministic Turing machines that read an input symbol at every step) and the quasi-real-time languages (the class of languages accepted by nondeterministic Turing machines that read an input symbol at every step). Kintala and Wotschke [10] counted the number of nondeterministic states an automaton passes through on its way to acceptance of an input. They were able to develop a ''succinctness'' hierarchy which measured the savings in the number of states of a nondeterministic finite automaton with more nondeterministic states over a nondeterministic finite automaton with fewer nondeterministic states relative to the corresponding minimal deterministic finite automaton. By using circuits as our basic model we have a means to easily quantify nondeterminism, and we extend the idea of limiting the amount of nondeterminism available to a computational device by developing a nondeterministic version of NC.

To define new complexity classes, we develop nondeterministic NC (NNC) circuits by uniformly adding ``*guessing gates*'' to families of LOGSPACE uniform NC circuits. We define NNC(f(n)) to be the class of sets accepted by LOGSPACE uniform families of NC circuits with at most O(f(n)) nondeterministic gates or guess gates, where n is the length of the input. We will refer to a circuit from such a family as a uniform NNC circuit, and such a circuit accepts an input if it outputs a 1. (See Cook [3] and Ruzzo [13] for complete descriptions of uniform circuit families.) The guessing gates give the circuit a set of guessing inputs, y, in addition to the ordinary inputs, x. An NNC circuit is said to accept x if and only if there is some string of guessing bits y that causes the circuit to output a 1. Thus f(n) can be thought of as the maximum number of guess bits used in computations on inputs of length n. The classes we study in this paper are of the form NNC($\log^k n$) and NNC(n^k). We will often abuse notation and write NNC(*class*) where *class* is a class of functions. For example, we define NNC(polylog) = $\bigcup_{k\geq 1}$ NNC($\log^k n$) and NNC(poly) = $\bigcup_{k\geq 1}$ NNC(n^k). We also refine the NNC($\log^k n$) classes to include an index that indicates the depth of the circuit. We let NNC^k($\log^j n$) denote the class of languages accepted by an NC^k circuit with O($\log^j n$) guessing gates, where an NC^k circuit is a circuit with depth O($\log^k n$). This definition is consistent with the notation in Cook [3] in that the exponent of NC indicates the depth of the circuit.

Note that NNC complexity classes share an intimate relationship with RNC, a random version of NC. RNC circuits and NNC circuits can be thought of as computing in exactly the same manner each with a different acceptance criteria. An NNC circuit accepts if and only if there is at least one string of guess bits that causes the circuit to output a 1, whereas an RNC circuit accepts if and only if at least ¾ of the possible strings of random (or guess) bits cause the circuit to output a 1. Unfortunately, RNC circuits are allowed to use a polynomial number of random bits, and thus, we can only say that RNC $\subseteq$ NNC(poly), which is not surprising since later we will show that NNC(poly) = NP. On the other hand, we know that any problem that has an RNC algorithm using O(f(n)) random bits is in NNC(f(n)).

One of our first goals is to determine how much nondeterminism is needed by an NC circuit in order for the associated complexity class to be new and "interesting." In Section 2 we will see that NNC(polylog) lies between NC and NP with neither containment known to be proper.

The parallel computation thesis states that parallel time is roughly equivalent to sequential space [2]. In Section 3 we explore an expansion of this idea by demonstrating some relationships between well-known DSPACE classes and the nondeterministic NC classes developed in Section 2. We are interested in the relationship between sequential space and the amount of nondeterminism given to NC circuits. Unfortunately, we are unable to show equivalence between the DSPACE classes and the NNC classes. In fact, we give some evidence that certain DSPACE classes and certain NNC classes are probably not equivalent.

In Section 4 we develop nondeterministic NC algorithms for the quasigroup isomorphism problem and related problems, problems not known to be in P or NP-complete. By applying the relationships developed in Section 3, we demonstrate that quasigroup isomorphism, Latin square isotopism and Latin square graph isomorphism are decidable in DSPACE($\log^2 n$). Our result, however, relies mostly on a new representation we develop for elements of quasigroups. This representation coupled with Miller's observation that a quasigroup has at most $\log_2 n$ generators leads to a DSPACE($\log^2 n$) algorithm as well as our NNC^2($\log^2 n$) algorithm for the quasigroup isomorphism problem.

We now describe the computational models used to define the usual nondeterministic and deterministic time and space complexity classes.

Our Turing machine models are familiar ones, and we assume the reader is familiar with Turing machines as described in Chapters 7 and 12 of Hopcroft and Ullman [7]. When we consider nondeterministic time bounds, our Turing machines have a fixed number of two-way infinite work tapes, one of which contains the input. For a nondeterministic machine, M, we say that a language L is accepted in time $T(n)$ by M if and only if on all inputs $w \in L$ of length n no sequence of choices causes M to make more than $T(n)$ moves. When considering deterministic space bounds, we often are concerned with bounds that are less than linear. In this case we adopt the ''off-line'' Turing machine with a two-way read only input tape and a fixed number of infinite work tapes. For a deterministic machine M, we say that a language L is accepted in space $S(n)$ by M if and only if on all inputs $w \in L$ of length n there is a valid computation leading to an accepting state that uses no more than $S(n)$ tape cells. Then L is said to be in DSPACE($S(n)$).

SECTION 2. Some Basic Relationships.

We would like to find a class of functions C such that NC $\subsetneqq$ NNC(C) $\subsetneqq$ NP. We begin by considering the class NNC(poly). We will show that this class is too powerful since NNC(poly) = NP. This is not surprising since Fortune and Wyllie [6] using a PRAM model for NC showed a similar result. Dymond [4] using a nondeterministic version of a hardware modification machine also showed that NP and nondeterministic NC are equivalent. The advantage of the NC circuit model has over these previous models is the straightforward manner in which the amount of nondeterminism used in computations can be quantified. Thus, next we consider the function $\log n$ and find that NNC($\log n$) is too weak, since NNC($\log n$) = NC. The most interesting case is when C is the class of functions that are bounded by polynomials in $\log n$. In Section 4 we show a form of quasigroup isomorphism is in NNC(polylog). Since this problem is not known to be in P and this problem is not known to be NP-complete, the class NNC(polylog) joins P, RP, ZPP and RNC as candidates for a class separating NC and NP.

We begin by considering the class NNC(poly). It is not hard to see that NNC(poly) $\subseteq$ NP.

Lemma 2.1: NNC(poly) $\subseteq$ NP.

Proof. Assume the set A is in NNC(poly). Then given x, an input of length n, and a description of the NNC circuit to accept words in A of length n, it is easy to construct an NP-machine to simulate the NNC circuit. First the NP machine guesses the outputs of the guess gates, and then since there are only a polynomial number of gates in the circuit, the NP machine can compute the output of each gate including the output gate in polynomial time. □

The next goal is to show NP $\subseteq$ NNC(poly), which implies NNC(poly) = NP. The idea of the proof is to take a nondeterministic one-tape Turing machine, M, that accepts a set A in NP and construct an NNC(poly) circuit, C_n that accepts inputs of length n. On input x of length n, C_n "guesses" an instantaneous descriptor (ID) for each move of an accepting computation of M on input x. An ID is a string of bits representing the contents of the tape, the position of the tape head (written in binary), and the current state of the machine M. Then for each pair of adjacent ID's a small circuit verifies that the second follows from the first via a legal move of M, based on the head position and the state of M. Furthermore, if each pair of adjacent ID's represent legal moves and the state of the last ID is an accepting state the circuit accepts, otherwise it rejects.

Theorem 2.2: NP = NNC(poly)

Proof. Lemma 2.1 shows one direction of the theorem. We now give more details of the argument outlined above.

Let M be a nondeterministic polynomial time bounded Turing Machine accepting a language L in NP. Without loss of generality it can be assumed M has only one tape and that every computation on inputs of length n takes q(n) steps, where q is some polynomial. Now consider a computation of M on input x of length n to be a table of instantaneous descriptors, where each ID represents the string currently on the work tape, the position on the tape of the read/write head written in binary, and the state of the machine (also in binary). Also, for the table to represent an accepting computation, ID_1 represents the initial state of the Turing machine, ID_i must follow from ID_{i-1} via a transition rule of M, and the last ID, $ID_{q(n)}$, must represent a final state of M if and only if $x \in L$.

To simulate the computation of M on x, C_n first guesses every ID of an accepting computation of M. There are O(q(n)^2) bits to guess since the length of each ID is O(q(n)) and there are q(n) ID's. Then for each pair of adjacent ID's there is a small circuit for testing whether ID_i follows from ID_{i-1} via a legal move of M. If ID_1 is correct, each ID follows from the previous one, and the state of $ID_{q(n)}$ is a final state, then the circuit accepts, otherwise it rejects.

The circuit used to verify that adjacent ID's represent legal moves of M is easy to construct as well. In parallel, it verifies that all the bits have remained the same from ID_{i-1} to ID_i except those in the area of the read/write head. In the area of the read/write head, a constant size circuit verifies that the move from ID_{i-1} to ID_i is a legal move of M by looking it up in a table.

Verifying ID_1 represents the initial state of M and verifying the state of $ID_{q(n)}$ is a final state are straightforward, and the detailed construction of these circuits is left to the reader.

Since these circuits can be uniformly constructed, we find that for each set in NP there is a uniform family of NNC(poly) circuits that accepts it. $\square$

Since NNC(poly) = NP, we need to consider functions other than polynomial ones to find a new and interesting nondeterministic version of NC. As the next theorem indicates, allowing log n guess bits does not allow the circuit to compute anything not in NC.

Theorem 2.3: $\mathrm{NNC}^k(\log n) = \mathrm{NC}^k$.

Proof. It is obvious that $\mathrm{NC}^k \subseteq \mathrm{NNC}^k(\log n)$. We now prove the other direction. Since there are only $2^{O(\log n)}$ (which is at most a polynomial in n) possible different guesses that can be made by the guessing gates, the circuit enumerates all of the possible guesses, and with duplicate copies of the NC circuit computes in parallel on each possible guess, accepting if at least one of the copies accepts. The circuit is still of polynomial size, and the depth of the circuit is increased only by an additional O(log n). □

Since O(log n) guess bits are too weak and polynomially many guess bits are too powerful, we settle on the class NNC(polylog) as potentially interesting as it is easy to see that NC $\subseteq$ NNC(polylog) $\subseteq$ NP. We fall short of our goal, though, since none of the containments are known to be proper.

SECTION 3. Relationships between NNC classes and DSPACE classes.

The first result in this section establishes a relationship between NNC classes and DSPACE classes and is instrumental in our result of the next section that the quasigroup isomorphism question can be decided in DSPACE($\log^2 n$). Lemma 3.1 shows that NNC(polylog) circuits that are deep and require little guessing and NNC(polylog) circuits that are shallow and require much guessing can both be simulated by a deterministic polylog-space bounded Turing machine. Using the more refined definitions of NNC complexity classes, this result can be stated more formally as the following lemma.

Lemma 3.1: For all $k,j \geq 1$, $\mathrm{NNC}^k(\log^j n) \subseteq \mathrm{DSPACE}(\log^m n)$, where $m = \max\{k,j\}$.

Proof. The deterministic Turing machine that simulates the NNC circuit begins by counting the number, N, of nondeterministic gates in the circuit. The Turing machine then writes the lexicographically first bit string, B, of length N on a work tape. Using a technique presented by Borodin [1] the Turing machine simulates the circuit by recursively evaluating first the left input and then the right input of a given gate, starting with the output gate. (I.e, the Turing machine does a depth first search of the circuit starting at the output gate.) At any time the Turing machine need only keep track of the status of each gate (which input is being computed and the value of the other input if it is known) on a path from the output gate to an input gate or a nondeterministic gate. If the name of a gate is ever needed, it can be recomputed using the status of each gate on the path starting with the output gate. Note that we store only a constant amount of information for each gate on the path and that we have at most one path active at any time. We continue this recursive procedure until a nondeterministic gate is encountered, at which point the computation is temporarily stopped. The name of this gate is written on a work tape, then M counts the number of gates, k, that precede it in the circuit description. Next M retrieves bit $k+1$ of B and uses it as the output of the current nondeterministic gate. If the circuit accepts, the Turing machine accepts, otherwise the Turing machine increments B and repeats the process until all the bit strings of length N have been tested.

For $k \geq j$, the depth of the circuit is larger than the number of nondeterministic bits, therefore the Turing machine is given enough space to simulate the circuit. For $j \geq k$, the bit string requires more space than does the simulation of the circuit, so the Turing machine is given enough space to write down the bit string. In both cases, the Turing machine has enough space to complete both tasks. □

Corollary 3.2 is an obvious consequence of Lemma 3.1. Since NC = $\bigcup_{k \geq 1} \mathrm{NC}^k$, clearly NNC(polylog) = $\bigcup_{k \geq 1} \bigcup_{j \geq 1} \mathrm{NNC}^k(\log^j n)$, and if we let DSPACE(polylog) = $\bigcup_{k \geq 1} \mathrm{DSPACE}(\log^k n)$ the following containment becomes apparent.

Corollary 3.2: NNC(polylog) $\subseteq$ DSPACE(polylog).

It would be desirable to show that either NNC(polylog) = DSPACE(polylog) or that NNC(polylog) $\subsetneqq$ DSPACE(polylog). With these goals in mind, we present the following lemma which begins to explore similar relationships between NNC(poly) and DSPACE(poly).

Lemma 3.3: For all k, NNC(n^k) $\subseteq$ DSPACE(n^k).

Proof. Since n^k grows much faster than any polylog function, use the construction of Lemma 3.1. □

Of course, to complete our expansion of the parallel computation thesis to the relationship between sequential space and parallel nondeterminism, it is desirable to show that for some k, DSPACE(n^k) $\subseteq$ NNC(n^k). However, that this containment is unlikely as the next lemma and theorem indicate.

Lemma 3.4: If there exist integers k and j such that DSPACE(n^k) $\subseteq$ NNC(n^j) then for all r there is an integer s such that DSPACE(n^r) $\subseteq$ NNC(n^s).

Proof. The proof of this claim is via the familiar translation technique (see [6]).

Let L_1 be any language in DSPACE(n^r), and let M_1 be an n^r space-bounded Turing machine accepting L_1. Let # be a symbol not in the alphabet of L_1, and let $L_2 = \{x\#^i \mid M_1$ accepts x using $(|x| + i)^k$ space$\}$. Thus we can construct a Turing machine M_2 such that on input $x\#^i$, M_2 marks off $(|x| + i)^k$ tape cells and then simulates M_1. M_2 accepts $x\#^i$ if and only if M_1 accepts x, and it does so using $(|x| + i)^k$ tape cells. Thus L_2 is in DSPACE(n^k), and by hypothesis L_2 is in NNC(n^j). Let C_2 denote the NNC(n^j) circuit that accepts the input $x\#^i \in L_2$.

Next we describe an NNC(n^s) circuit, C_1, that accepts words in L_1 of length n. On input x of length n, C_1 guesses an i. Next C_1 simulates C_2 on $x\#^i$, and C_1 accepts if and only if C_2 accepts. Since $|i|$ is O(n^r) and C_2 guesses $O((n+i)^j) = O(n^{rj})$ bits, we let $s = rj$. Also note that the depth of C_1 is no more than that of C_2 and that the number of gates in C_2 (and thus C_1) is polynomial in $|x|^r$ and thus polynomial in $|x|$. Therefore we have an s such that DSPACE(n^r) $\subseteq$ NNC(n^s). □

Theorem 3.5 presents strong evidence that the containment in Lemma 3.3 is proper.

Theorem 3.5: If there exist k and j such that DSPACE(n^k) $\subseteq$ NNC(n^j) then PSPACE = NP.

Proof. From Theorem 2.2 we know that NP = NNC(poly) = $\bigcup_{k \geq 1}$ NNC(n^k), and by definition PSPACE = $\bigcup_{k \geq 1}$ DSPACE(n^k). Since Lemma 3.4 under the same hypothesis as this theorem states that for all r there is an s such that DSPACE(n^r) $\subseteq$ NNC(n^s), we have PSPACE $\subseteq$ NNC(poly) = NP, giving the desired result. □

Theorem 3.5 provides some information about whether NNC(n^k) is properly contained in any DSPACE(n^j) class. It would be desirable to find similar information concerning the containment of NNC($\log^k n$) classes in DSPACE($\log^j n$) classes. Doing so, however, seems unlikely since DSPACE($\log n$) $\subseteq$ NC = NNC($\log n$). Thus to show a result such as "If there exist k and j such that DSPACE($\log^k n$) $\subseteq$ NNC($\log^j n$) then an unlikely collapse" would require developing a proof technique that forces the unlikely collapse for $k \geq 2$ but does not force the collapse for $k = 1$. Unfortunately, even the need for such a peculiar proof technique gives us little insight into whether or not NNC(polylog) is properly contained in DSPACE(polylog).

SECTION 4. Quasigroup isomorphism is in DSPACE($\log^2$ n).

In this section we show that it is worthwhile to consider NNC complexity classes by showing that a number of interesting problems lie in NNC^2($\log^2 n$). Since Miller [12] has shown the quasigroup (Latin square) isomorphism problem has an $O(n^{\log_2 n + O(1)})$ sequential algorithm, and since $n^{\log_2 n} = 2^{\log_2^2 n}$ this problem is a natural candidate for being in NNC^2($\log^2 n$). For review we give the definitions of Latin squares, groups and quasigroups.

Definition: A *Latin square* is an $n \times n$ grid with each of the integers $1, 2, \cdots, n$ appearing exactly once in each row and column.

If each of the integers $1, 2, \cdots, n$ appears as a label for exactly one row and exactly one column then the Latin square can be viewed as a multiplication table of a quasigroup. We formalize the definitions of groups and quasigroups by considering the following four properties of a set Q with an associated binary operation *. For all $a,\ b,\ c \in Q$:

1. There is a unique x such that $a*b = x$.
2. There is a unique x such that $a*x = b$.
3. There is a unique x such that $x*a = b$.
4. $(a*b)*c = a*(b*c)$.

Definition: Q is a *group* if * satisfies properties 1, 2, 3 and 4.

Definition: Q is a *quasigroup* if * satisfies properties 1, 2 and 3.

Thus a quasigroup is more general than a group. In this paper we view quasigroups of order n as a binary function on $\{1, 2, \cdots, n\}$, thus the corresponding multiplication table is a Latin square. Viewing Latin squares L and L' as trinary relations $< , , >$ and $< , , >'$, L is *isomorphic* to L' if there exists a permutation σ such that if $<x,y,z> \in L$ then $<\sigma(x),\sigma(y),\sigma(z)>' \in L'$. Two quasigroups are isomorphic if their corresponding Latin squares are isomorphic. A more general notion of an isomorphism is an isotopism. L is *isotopic* to L' if there exist permutations α, β, γ such that if $<x,y,z> \in L$ then $<\alpha(x),\beta(y),\gamma(z)>' \in L'$. Thus an isomorphism simultaneously interchanges rows, columns and values in L to get L', and an isotopism independently interchanges rows, columns and values in L to get L'. Miller [12] showed that quasigroups of order n are generated by at most $\log_2 n$ elements, and we take advantage of this fact to develop the circuit for quasigroup isomorphism testing, Latin square isotopism testing, and Latin square graph isomorphism testing. Since quasigroups are more general than groups, our construction also shows group isomorphism is also in $\mathrm{NNC}^2(\log^2 n)$. Note that we assume that quasigroups are input as their corresponding Cayley (multiplication) tables.

Before getting to the main results of this section we develop a useful representation for elements of quasigroups in terms of generators. The generating set is a subset of the quasigroup such that every element in the quasigroup can be expressed as a product of elements from that set. Since quasigroups are not necessarily associative, when an element $q \in Q$ is written as the product of generators from a particular generating set, it must be fully parenthesized, for example, $q = (((g_1 * g_2) * (g_3 * g_2)) * g_3)$. We associate with $q \in Q$ the parse trees of all such expressions with internal nodes representing multiplications and leaves representing generators. If q is in the generating set, then q is represented by a tree consisting of a single node labeled q. If q is not in the generating set and $q = p * r$, then q is represented by a tree where the root represents the multiplication between the element represented by the left subtree (p) and the element represented by the right subtree (r). This definition gives infinitely many representations for each element in Q, including some very large representations. If we can show that for every element in the quasigroup there is a tree of small depth[2] that represents it, then the large representa-

[2] The depth of a node is the number of edges between it and the root. The depth of a tree is the maximum depth of a node over all nodes in the tree.

tions will not hinder us.

To show that each element has a shallow representation we build a sequence of nonempty sets based on the depth of the shallowest tree representing each element and show that that sequence is not too long. Let depth(q) be the depth of the shallowest tree representing $q \in Q$. Since $Q = \{1, 2, \cdots, n\}$ we can also assume that for $i, j \in Q$, if $i \leq j$, then depth(i) $\leq$ depth(j). Let $depth_k = \{q \in Q \mid \text{depth}(q) \geq k\}$ be the set of all elements in Q with shallowest tree representation at least k. Note that $depth_0 = Q$. Since Q is finite, there is an integer d such that $n \in depth_d$ and for all integers $m > d$, $depth_m$ is empty. Now for the sequence of sets $depth_0 \supseteq depth_1 \supseteq \cdots \supseteq depth_d \supseteq depth_{d+1} = \emptyset$, we will demonstrate that each $depth_k$ is nonempty, that each of the containments in the sequence is proper and that d is not too large. For notational convenience, we let $Q_l = \{1, 2, \cdots, l\}$ for $1 \leq l \leq n$ and let Q_0 be the empty set. The following lemma establishes that each $depth_k$ is nonempty for each k, $0 \leq k \leq d$, and that each of the containments is proper. Essentially the lemma says for even h, $depth_{d-h}$ contains at least one element of Q not in $depth_{d-h+1}$, and, for odd h, $depth_{d-h}$ contains at least half the elements of Q that are not in $depth_{d-h+2}$.

Lemma 4.1: Let h be an even integer, $0 \leq h \leq d$, and let G be any generating set of Q, and let $k = |G|$. Then $|depth_{d-h-1}| \geq \sum_{i=1}^{\frac{h}{2}+1} \frac{n}{2^i}$, and either (1) there is an $r \in depth_{d-h-1}$ such that $r \notin depth_{d-h}$ and there exist $s, t \in Q - depth_{d-h-1}$ with $r = s * t$ with either s or t in $depth_{d-h-2}$ or (2) $G \subseteq depth_{d-h-1}$.

Proof. The proof is by induction on h. For the base case, let $h = 0$. By assumption, n is the maximal element of Q in $depth_d$. Since n appears once in each row and column of the multiplication table of Q at least half of the elements of Q must have depth at least $d - 1$ otherwise n

does not have depth d, thus $|depth_{d-1}| \geq n/2$. Partition Q into two parts, R = $depth_{d-1}$ = $\{m+1, m+2, \cdots, n\}$ and $Q_m = \{1, 2, \cdots, m\}$, and note that $m \leq n/2$ and that no element of Q_m has depth larger than $d-2$. This partition divides the multiplication table of Q into four sections, $Q_m \times Q_m$, $Q_m \times R$, $R \times Q_m$, and $R \times R$. If $k > m$, where k is the number of generators, then m = 0, i.e., there are no elements in Q in $depth_{d-2}$. Therefore $G \subseteq depth_{d-1}$. So assume $k \leq m$. Since all of the generators of Q are in Q_m, at least one element $r \in R$ must appear in section $Q_m \times Q_m$ of the multiplication table, otherwise G would not generate Q. Thus $r = s*t$, where s, $t \in Q_m = Q - depth_{d-1}$. At least one of s and t must be in $depth_{d-2}$ otherwise depth(r) < $d-1$. Certainly $r \notin depth_d$ otherwise s or t would be in $depth_{d-1}$, contradicting the fact that all elements of depth $d-1$ are in R.

For the inductive step, assume $h \geq 2$, and assume the lemma holds for h - 2. Let l be the maximal element of Q such that depth(l) = $d-h$. The inductive hypothesis also gives $r \in depth_{d-h+1}$ such that $r \notin depth_{d-h+2}$ and $r = s*t$ with either s or t in $depth_{d-h}$. Without loss of generality assume $s \in depth_{d-h}$. To complete the inductive step, we first digress and establish Claim 4.2.

Claim 4.2: In every row (column) of $Q_l \times Q_l$ there is an element $x \in depth_{d-h}$.

Proof. We know that $s \in depth_{d-h}$, and since $s \in Q_l$, $s \notin depth_{d-h+1}$. Assume the lemma does not hold, and let i be the label of the row (column) of $Q_l \times Q_l$ with no elements in $depth_{d-h}$. Now $|depth_{d-h+1}| = n - l$, and each element of $depth_{d-h+1}$ as well as s must appear somewhere in the i^{th} row (column) of the multiplication table of Q. Since $\{s\} \cup depth_{d-h+1} \subseteq depth_{d-h}$ and $|\{s\} \cup depth_{d-h+1}| = n - l + 1$, at least one element in $\{s\} \cup depth_{d-h+1}$ must appear as one of the first l elements of row (column) i, contradicting the assumption that none of the first l elements of row (column) i are in $depth_{d-h}$. □

Resuming the inductive step, we see that Claim 4.2 implies that at least $l/2$ elements of Q_l are in $depth_{d-h-1}$. We know $|depth_{d-h-1}|$ is at least the number of elements in Q_l that are in $depth_{d-h-1}$ plus the number of elements not in Q_l that are in $depth_{d-h-1}$. So $|depth_{d-h-1}| = |depth_{d-h-1} \cap Q_l| + |depth_{d-h-1} - Q_l| \geq l/2 + |depth_{d-h+1}|$, since $depth_{d-h-1} - Q_l = depth_{d-h+1}$. Since $Q_l = Q - depth_{d-h+1}$, we have $l = n - |depth_{d-h+1}|$. Thus $|depth_{d-h-1}| \geq ½(n + |depth_{d-h+1}|)$, and invoking the induction hypothesis, $|depth_{d-h-1}| \geq ½(n + \sum_{i=1}^{\frac{h}{2}} \frac{n}{2^i}) = \sum_{i=1}^{\frac{h}{2}+1} \frac{n}{2^i}$.

Now proceeding as in the base case, we partition Q into two parts, $Q_m = \{1,2,\cdots,m\}$ and $R = depth_{d-h} = \{m+1, m+2, \cdots, n\}$ and note that $m < l/2$ and no element of Q_m has depth greater than $d-h-2$. Using arguments similar to those of the base case we find that if $k > m$, then $G \subseteq depth_{d-h-1}$, and if $k \leq m$ we can find an r satisfying the conditions of Lemma 4.1. □

Now we can easily show that each element of the quasigroup is represented by a shallow tree.

Lemma 4.3: If Q is a quasigroup of order $n \geq 2$ and $G = \{g_1, g_2, \cdots, g_k\}$ generates Q, then for every $q \in Q$ there is a tree that represents it with depth no more than $2\log_2 n$.

Proof. Since the size of the generating set G is at least one, by Lemma 4.1 we need only find the smallest h such that $\sum_{i=1}^{\frac{h}{2}+1} \frac{n}{2^i} \geq n-1$. It is easy to show that for $h \leq 2\log_2 n - 2$, the above inequality holds. Thus $h \leq 2\log_2 n - 2$. Now fix h. Since $depth_1$ contains everything in Q except the generators, we have $depth_1 = depth_{d-h-1}$, which implies $h = d-2$, giving $d \leq 2\log_2 n$. Recalling that d is the depth of the deepest element, we have the desired result. □

With this shallow tree representation of elements of a quasigroup, we now can show that quasigroup isomorphism can be computed with an $\mathrm{NNC}^2(\log^2 n)$ circuit.

Theorem 4.4: Given two multiplication tables M_1 and M_2, representing the quasigroups Q_1 and Q_2, respectively, the set $\{(Q_1, Q_2) \mid Q_1$ is isomorphic to $Q_2\}$ is in NNC^2($\log^2 n$), where n is the order of the two quasigroups.

Proof. We first give a general overview of the circuit that tests for the isomorphism, and later we give a more detailed construction. The circuit to test the isomorphism begins by guessing two sets of generators G_1 for Q_1 and G_2 for Q_2 in parallel. We will assume that G_1 and G_2 are ordered in some manner and that order determines the isomorphism, i.e., the i^{th} element of G_1 is mapped to the i^{th} element of G_2. Next in parallel the circuit verifies that G_1 generates Q_1, G_2 generates Q_2, and that the mapping guessed is an isomorphism. Verifying G_i generates Q_i involves two general steps. First, elements that are known to be generated by the generators are marked as being in the quasigroup, with the guessed generators marked initially. Second, in parallel each marked element of the quasigroup is multiplied by every marked element of the quasigroup, generating more elements known to be in the quasigroup. To verify the guessed mapping is an isomorphism the circuit performs two tasks in parallel. First, in parallel for each pair of elements g, $h \in G_1$, the circuit tests whether g and h are the same, and if so, it verifies that their images in G_2 are identical. Second, in parallel for each pair of elements g, $h \in G_2$, the circuit tests whether g and h are the same, and if so, it verifies that their preimages in G_1 are identical.

We now provide more details of the isomorphism testing circuit. Miller [12] has shown that a quasigroup of size n has at most $\log_2 n$ generators, and we note that each generator is $\log_2 n$ bits long, thus the circuit has $2\log_2{}^2 n$ guessing gates, half of which are used to guess G_1, the set of generators of Q_1, and the rest are used to guess G_2, the set of generators of Q_2.

To verify that G_1 does generate Q_1 we use the following subcircuit with $2\log_2 n$ levels. An identical circuit will verify G_2 generates Q_2. Each level of the subcircuit corresponds to a copy of the multiplication table of Q_1, and the i^{th} level, $level_i$, computes all the elements of Q_1 that can be expressed as the product of at most 2^i elements of G_1. In order to do this, $level_i$ receives from $level_{i-1}$ all of the elements of Q_1 that can be expressed as the product of at most 2^{i-1} generators, and then in parallel multiplies each of those elements by every element it received from the previous level by looking up the product in the input multiplication table. Instead of receiving inputs from the previous level, $level_1$ receives inputs from the guessing gates, as the generators are the only elements of Q_1 that can be expressed as the product of at most one element of G_1. After the final level, a check is made to insure that all of the elements of Q_1 have been generated.

From Lemma 4.3 we know that each element in a quasigroup of order n has a tree with depth no more than $2\log_2 n$ representing it. Thus after $2\log_2 n$ levels either all of the elements are generated or G_i is not a generating set for Q_i, and therefore there are only O(log n) levels in the circuits that verify G_i generates Q_i. At each level, there are at most n^2 multiplications taking place, with each multiplication requiring an O(log n) depth circuit with polynomial size. Since there are O(log n) levels, we have an overall depth of O($\log^2 n$), and the overall circuit size is polynomial.

Verifying that the guessed mapping is an isomorphism is straightforward. Let $G_1 = \{g_1, g_2, \cdots, g_k\}$, and let $G_2 = \{h_1, h_2, \cdots, h_k\}$. The circuit verifies for all $1 \le i,j \le k$ if $g_i = g_j$ then $h_i = h_j$, and if $h_i = h_j$ then $g_i = g_j$. Since there are $\log_2 n$ elements in each G_i, there are ${\log_2}^2 n$ pairs that must be tested. Each test can be computed by an O(loglog n) depth and O(log n) size circuit. Thus quasigroup isomorphism is in NNC^2($\log^2 n$). □

If we let L be the Latin square associated with Q_1 and L' be the Latin square associated with Q_2 in the previous proof, then we can view the guessing of the generating sets as the guessing of a permutation σ that for all $x,y,z \in G_1$ takes triples $<x,y,z> \in L$ to triples $<\sigma(x),\sigma(y),\sigma(z)>' \in L'$. Now σ can also be applied to elements of Q_1 not in G_1 if they first are written as the products of generators. To show Latin square isotopism is in $\mathrm{NNC}^2(\log^2 n)$ we guess three permutations α, β, γ from three generating sets of L to three generating sets of L' and show that if $<x,y,z> \in L$, then $<\alpha(x),\beta(y),\gamma(z)>' \in L'$.

Theorem 4.5: Given two Latin squares L and L' as lists of triples of the form $<a,b,c>$ and $<x,y,z>'$, respectively, then the set $\{(L, L') \mid L \text{ is isotopic to } L'\}$ is in $\mathrm{NNC}^2(\log^2 n)$, where n is the size of the two Latin squares.

Proof. To show two Latin squares are isotopic we need three surjective functions between L and L'. The circuit begins by guessing subsets $A, B \subset L$ and subsets $A', B' \subset L'$. Let $A = \{a_1, a_2, \cdots, a_k\}$, $B = \{b_1, b_2, \cdots, b_k\}$, $A' = \{a'_1, a'_2, \cdots, a'_k\}$ and $B' = \{b'_1, b'_2, \cdots, b'_k\}$, where $k = \lfloor \log n \rfloor$. Let $C = \{a_1 * b_1, a_2 * b_2, \cdots, a_k * b_k\}$, and $C' = \{a'_1 *' b'_1, a'_2 *' b'_2, \cdots, a'_k *' b'_k\}$, where $*$ and $*'$ are the appropriate binary operators for the respective quasigroups. If we show that A, B and C each generate L and that A', B' and C' each generate L', then we can find the surjective maps needed to take L to L' as follows:

α: For $x \in L$, if $x = a_i$ then $\alpha(x) = a_i$. Else if $x = y * z$ then $\alpha(x) = \alpha(y) *' \alpha(z)$.

β: For $x \in L$, if $x = a_i$ then $\beta(x) = b_i$. Else if $x = y * z$ then $\beta(x) = \beta(y) *' \beta(z)$.

γ: For $x \in L$, if $x = a_i * b_i$ then $\gamma(x) = a_i * b_i$. Else if $x = y * z$ then $\gamma(x) = \gamma(y) *' \gamma(z)$.

Thus we need only test that each of α, β and γ is consistent on A and A', B and B', and C and C', respectively. Using a circuit similar to the one used in deciding quasigroup isomorphism we find that Latin square isotopism can be decided by a circuit with $O(\log^2 n)$ guessing gates and $O(\log^2 n)$ depth. Thus Latin square isotopism is in $\mathrm{NNC}^2(\log^2 n)$. □

Latin squares give rise to a special class of graphs called Latin square graphs. A Latin square graph consists of n^2 nodes, one corresponding to each of the triples of the Latin square. Two nodes $<x,y,z>$ and $<u,v,w>$ are adjacent if $x=u$, $y=v$ or $z=w$. Namely, two nodes are adjacent if they are in the same row or column of the Latin square or if they share the same value. Thus Latin square graphs of size n consist of $3n$ n-cliques.

To show Latin square isomorphism is in NNC$^2(\log^2 n)$ we need another notion of isotopism. Two Latin squares L and L' are *conjugate* if $<x_1, x_2, x_3> \in L$ implies $<x_{\alpha(1)}, x_{\alpha(2)}, x_{\alpha(3)}>' \in L'$, where α is a permutation in S_3. L and L' are *main class isotopic* if we can get from L to L' by a conjugation and an isotopic map. Since there are only six permutations in S_3, main class isotopism can be decided in NNC$^2(\log^2 n)$ by giving the circuit that decides isotopism the ability to guess which one of the six permutations in S_3 to use. The next result from Miller [11] gives the relationship between Latin squares and Latin square graphs.

Lemma 4.6 [11]: Let L and L' be two Latin squares and G(L) and G(L') be the associated Latin square graphs. L is main class isotopic to L' if and only if G(L) is isomorphic to G(L').

In Lemma 4.7 we give a means to retrieve the Latin square from a Latin square graph with a circuit with depth no more than O($\log^2 n$) depth. The algorithm given is the obvious parallelization of Miller's sequential algorithm that does the same thing in O(n^3) sequential time.

Lemma 4.7: We can retrieve the Latin square from the Latin square graph with a polynomial sized circuit with depth less than O($\log^2 n$).

Proof. Let $L(l_{ij})$ be an $n \times n$ matrix used to store the Latin square.

(1) Pick two adjacent nodes x_1 and x_2.

(2) In parallel find the n nodes adjacent to both x_1 and x_2. All but two of the nodes form an n-clique with x_1 and x_2. In parallel, label each node in the clique $x_3, \cdots, x_n$. One node not adjacent to any of $x_3, \cdots, x_n$ is labeled y_2.

(3) Associate m_{1j} with x_j, and in parallel set m_{1j} to j.

(4) In parallel find the clique associated with x_1 and y_2, $\{x_1, y_2, y_3, \cdots, y_n\}$.

(5) Each x_i shares an edge with some y_j, $2 \le i, j \le n$. Order the y_j's so that x_j shares an edge with y_j.

(6) Associate m_{j1} with y_j, and in parallel set m_{j1} to j for $2 \le j \le n$.

(7) In parallel for each of the $(n-1)^2$ remaining nodes z of the graph:
   a) If z is adjacent to x_1 then z is adjacent to a unique y_i and a unique x_j, $2 \le i, j \le n$. Set m_{ij} to 1.
   b) Else z is not adjacent to x_1, and there are unique integers i, j and k such that z is adjacent to x_i, y_j, x_k, and y_k. Set m_{ij} to k.

Each of the steps can be computed by an O(log n) depth circuit with a polynomial number of gates, thus the Latin square can be easily retrieved from the Latin square graph. □

Theorem 4.8: Given to Latin square graphs G_1 and G_2, the set $\{(G_1, G_2) \mid G_1$ is isomorphic to $G_2\}$ is in $\mathrm{NNC}^2(\log^2 n)$, where n is the size of the associated Latin squares.

Proof. This follows directly from Theorem 4.5, Lemma 4.6, Lemma 4.7 and the definition of main class isotopism. □

We now reach the main result of this section.

Theorem 4.9: Quasigroup isomorphism, Latin square isotopism and Latin square graph isomorphism are in DSPACE($\log^2 n$).

Proof. This follows directly from Theorem 4.4, Theorem 4.5, Theorem 4.8 and Lemma 3.1. □

SECTION 5. Conclusion.

Initially, in our study of nondeterministic complexity classes our goal was to determine how much nondeterminism an NC circuit should be given to cause "interesting" things to happen. By considering NC circuits with a polylog number of guessing gates we were able to show that quasigroup isomorphism is in DSPACE($\log^2 n$). We also saw that NNC(polylog) is a potential candidate for separating NP from NC and that NNC(n^k) is probably different from NNC(n^j) for all j and k. Some open problems include exploring the relationship between P and NNC(polylog) as well as exploring the relationship between Random NC (RNC) and NNC(polylog). We conjecture that NNC(polylog) and P are incomparable. The fact that P-complete problems cannot be easily decided by NNC(polylog) circuits suggests P $\not\subset$ NNC(polylog). Since group and quasigroup isomorphism are not known to be in P, we have evidence that NNC(polylog) $\not\subset$ P. It would be interesting to determine if NNC(polylog) $\cap$ P = NC.

We also suspect that RNC and NNC(polylog) are also incomparable, although the evidence is not as clear. In some sense, RNC and NNC(polylog) are very different complexity classes. For a set to be in RNC there must be many (a polynomial number) polynomial length witnesses for every string in the set. On the other hand, a string in an NNC(polylog) set needs only one witness, and that witness can be short—it only needs to be of polylog length. We do have some weak evidence that RNC $\neq$ NNC(polylog). We note that using the obvious approach, quasigroup isomorphism cannot be shown to be in RNC. If the two input quasigroups are cyclic groups of order n it is not difficult to show that fixing the mapping between one pair of generators fixes the mappings between all of the remaining pairs of generators. This forces the probability of finding a string that encodes an isomorphism between the two groups to be less than $\frac{1}{O(n^{\log n})}$ which is certainly less than $1/p(n)$.

A final open problem worth considering is determining the intersection of RNC and NNC(polylog). In light of the recent result of Berger and Rompel [1] that ($\log^k n$)-wise independence can be simulated in NC, it seems reasonable to suspect that RNC $\cap$ NNC(polylog) = NC.

REFERENCES

[1] B. Berger and J. Rompel, Simulating ($\log^c n$)-wise independence in NC, to appear in *Proceedings of the Thirtieth Annual IEEE Symposium on the Foundations of Computer Science* (1989).

[2] A.B. Borodin, On relating time and space to size and depth, *SIAM Journal of Computing* 6 (1975) 733-744.

[3] S.A. Cook, The classification of problems which have fast parallel algorithms, *Lecture Notes in Computer Science* V. 158, Springer-Verlag, New York, (1983) 78-93.

[4] P.W. Dymond, On nondeterminism in parallel computation, *Theoretical Computer Science* 47 (1986) 111-120.

[5] P.C. Fischer and C.M.R. Kintala, Real-time computations with restricted nondeterminism, *Mathematical Systems Theory* 12 (1979) 219-231.

[6] S. Fortune and J. Wyllie, Parallelism in random access machines, *Proceedings of the Tenth ACM Symposium on the Theory of Computing* (1978) 114-118.

[7] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation*, (Addison-Wesley, Reading, Mass., 1979).

[8] C.M.R. Kintala and P.C. Fischer, Computation with a restricted number of nondeterministic steps, *Proceedings of the Ninth ACM Symposium on the Theory of Computing* (1977) 178-185.

[9] C.M.R. Kintala and P.C. Fischer, Refining nondeterminism in relativized polynomial-time bounded computations, *SIAM Journal of Computing* 9 (1980) 46-53.

[10] C.M.R. Kintala and D. Wotschke, Amounts of nondeterminism in finite automata, *Acta Informatica* 13 (1980) 199-204.

[11] R.J. Lipton, L. Snyder and Y. Zalcstein, Complexity of the word and isomorphism problems for finite groups, *Proceedings of the Conference on Information Sciences and Systems*, 10 (1976) 33-35.

[12] G.L. Miller, On the $n^{\log n}$ isomorphism technique, *Proceedings of the Tenth ACM Symposium on the Theory of Computing* (1978) 51-58.

[13] W.L. Ruzzo, On uniform circuit complexity, *Journal of Computer and System Sciences*, 22 (1981) 365-383.