Non-monotone Behaviors in MIN/MAX/AVG Circuits and their Relationship to Simple Stochastic Games[*]

Manuel Blum, Brendan Juba, Ryan Williams

Abstract

In this paper, we examine the problem of finding optimal strategies in *simple stochastic games*, and the equivalent problem of finding stable configurations of MIN/MAX/AVG circuits. The problem can be seen as a nontrivial extension to linear programming, but no polynomial-time algorithm is known, despite significant efforts to find such an algorithm. Our investigation will be concerned with the relative computational power of properties of these circuits. We will begin by defining the computational problems associated with these games and circuits, and stating in which senses the two are equivalent. We observe that the stable circuit problem is in **PLS**$\cap$**PPAD** and will demonstrate the power of MIN/MAX/AVG circuits when we allow the circuit to signal acceptance by switching the direction of output of a MIN or MAX gate. Specifically, we find that the associated computational problem is **PSPACE**-hard, so unless **NP** = **PSPACE** this variant is harder than the original stable circuit problem.

1 Introduction

Simple stochastic games are a class of games originally introduced and studied by Shapley in 1953 [12]. The computational problem of determining whether or not a player has a probability of winning a simple stochastic game of greater than 1/2 was studied extensively by Condon much later [5]. At that point, the problem was widely conjectured to be solvable in polynomial-time, but many early attempts at algorithms were either shown to be incorrect or have known exponential counterexamples [6]. Since then, significant effort has been put into finding algorithms with little success, so we conjecture that the problem is neither in **P** nor in **BPP**, and thus we wish to show that the problem is hard for some natural class.

Of course, since the problem has been shown to be in **NP**$\cap$**coNP** [5], we have restricted our attention to classes relatively close to **P**. This suggests examining classes such as **PLS** or **PPAD** for hardness results – we see in Section 3 that an equivalent function problem is contained in both – but as we will discuss later, hardness results may not be possible. Certainly, if it is hard for one of these classes, it will have some interesting consequences, as we explore in Section 4.

To get a handle on the hardness of simple stochastic games, we were interested in seeing what sorts of gadgets we could build out of instances of simple stochastic games. One of the interesting features of this problem is that it has a variety of equivalent formulations. Condon [5] observed that the problem is equivalent to the decision problem for languages accepted by logspace alternating randomized Turing machines. In this paper, in Section 2, we exhibit yet another natural equivalent formulation of the problem: finding the stable configurations of circuits of MIN, MAX, and AVG gates with fixed 0/1 inputs. In particular, we show that an instance of one yields an instance of

[*]`mblum@cs.cmu.edu`, `bjuba@mit.edu`, `ryanw@cs.cmu.edu`. Carnegie Mellon University 5000 Forbes Ave. Pittsburgh, PA 15213. This work was supported in part by the National Science Foundation as part of the Aladdin Center (`www.aladdin.cmu.edu`) under grant CCR-0122581, and associated REU funding.

the other in quite a natural way, and that the computational problems associated with each are polynomial-time equivalent. This result is not new; rather, it is a bit of folklore which we have fleshed out [1]. It was then using this circuit formulation of the problem that we began attempting to build gadgets.

In particular, we define a variant on the problem, examining the circuits under a "Leapfrog" model, in which we allow ourselves to read a bit of information from the MIN or MAX gates—we informally say that the circuit accepts if a particular gate chooses a different input (the LEFT vs. the RIGHT input) during its computation. We present a more formal definition and a brief discussion of the motivation behind this definition in Section 2.4. We had hoped to achieve the following:

1. Find a suitable hardness result for "Leapfrog" circuits
2. Reduce the "Leapfrog" model to the usual stable circuit problem

our work toward these ends is the focus of Section 3, where we show that these circuits can solve **PSPACE**-hard problems (specifically TQBF); along the way, we give an approach to developing a binary counter in an analog monotone circuit, a seemingly counter-intuitive feat. As such, while these "Leapfrog" circuits are interesting in their own right, they are probably not suitable for use in a reduction to some class contained in **NP**$\cap$**coNP** , under standard complexity-theoretic assumptions. Section 3.3 defines an alternative decision problem for Leapfrog circuits, which is also **PSPACE**-hard for much the same reasons, providing evidence that suggests that the second goal described above cannot be achieved.

We wrap up by discussing all of these considerations in greater detail in Section 4, particularly emphasizing the difficulties we encountered in designing gadgets in MIN/MAX/AVG circuits. We further develop the relationships betweeen the stable circuit problem and the classes **PLS** and **PPAD**, and finally close by suggesting some interesting open questions raised by this work.

2 Definitions and Preliminaries

Sections 2.1 and 2.2 introduce the usual definitions for simple stochastic games and MIN/MAX/AVG circuits, respectively. We define our simple stochastic games as done by Condon [5] and define our MIN/MAX/AVG circuits largely as in [2]. In Section 2.3, we make explicit the implicit equivalences hinted at in [2], using some preliminary results from Condon [5]. Finally in Section 2.4, we formally state a new variant on the problem which we will focus on, and in Section 2.5, we restate one important result due to Condon [5], that the simple stochastic game problem is in **NP**$\cap$**coNP**, state a claim implicitly developed by Condon [5] and used in [2], and state one important result due to Papadimitriou: that a computational version of Brouwer fixed-points is in his class **PPAD** [10].

2.1 Simple Stochastic Games

Supppose we have a directed graph $G = (V, E)$ in which each vertex has outdegree of at most two, with the exception of two special vertices, $\mathbb{O}, \mathbb{I} \in V$ called the *0-sink* and the *1-sink*, respectively, each of which have out-degree 0. Further suppose G has its vertex set partitioned as follows:

$$V = MIN \cup MAX \cup AVG \cup \{\mathbb{O}\} \cup \{\mathbb{I}\}$$

A *simple stochastic game* is played on this graph by two players, known as the *0-player* and the *1-player*. A token is placed on a designated *start vertex*, *start* $\in V$, and it is moved across the

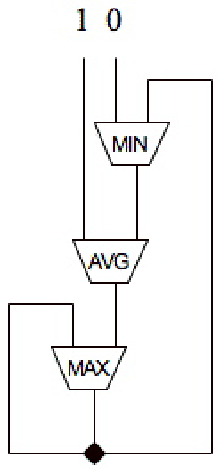

Figure 1: Example MIN/MAX/AVG circuit

the edges of the graph in the following manner: if the token is at $v \in MAX$, the 1-player chooses which of the neighbors of v to move the token across to. Similarly, from $v \in MIN$, the 0-player chooses which neighbor of v to move the token to, while at $v \in AVG$, the two players flip a fair coin to decide which neighbor will be moved to. The objective of the 1-player is to reach the 1-sink, whereas the objective of the 0-player is simply to confound the 1-player (e.g. either by arriving at the 0-sink *or* by keeping the vertex moving in some loop).

We may consider a *strategy* of some player to be a subset of the edges; specifically, a strategy of the 1-player is defined by $\sigma \subseteq E$ containing for each $v \in MAX$ precisely one edge out of v, and strategies for the 0-player are defined similarly, contining one edge for each $u \in MIN$. From a vertex $v \in MAX$, the 1-player chooses to move across the unique $e \in \sigma$ out of v.

Given a pair of strategies, σ and τ we can assign a *value* to each vertex, $value(v, \sigma, \tau)$, the probability that the 1-sink is reached from v given the 1-player plays by σ and the 0-player plays by τ. We can now define the *value of the game* starting from $start \in V$ to be

$$value(start) = \max_{\sigma} \min_{\tau} value(start, \sigma, \tau)$$

e.g., the probability that the 1-player wins in an optimal strategy against a best-response of the 0-player. These values were studied extensively by Condon [5], and we follow her definition of the problem SSG-VALUE($G, start$): "in the simple stochastic game on G, is $value(start) \geq 1/2$?"

2.2 MIN/MAX/AVG Circuits

Consider a circuit C of MIN/MAX/AVG gates, with feedback *allowed*, each gate having fan-in two, and *inputs* to the circuit hard-wired to either constant 0 or constant 1. (see figure 1) We let $|C|$ denote the number of gates in C, and we assume that the output of each gate is some real number from the closed interval $[0, 1]$. We will frequently discuss *value vectors*, $\vec{x} \in [0, 1]^{|C|}$, which represent a setting for each gate output—$x_i \in [0, 1]$ denotes the output of the ith gate. Notice that this is an implicit ordering of the gates of C. We *do not* require that the output of the ith gate be the respective function of its inputs, but when this is so, e.g. if the ith gate is a MAX gate with inputs from gates j and k, and $x_i = \max\{x_j, x_k\}$, then we say that the gate is *satisfied* by the assignment $\vec{x}$.

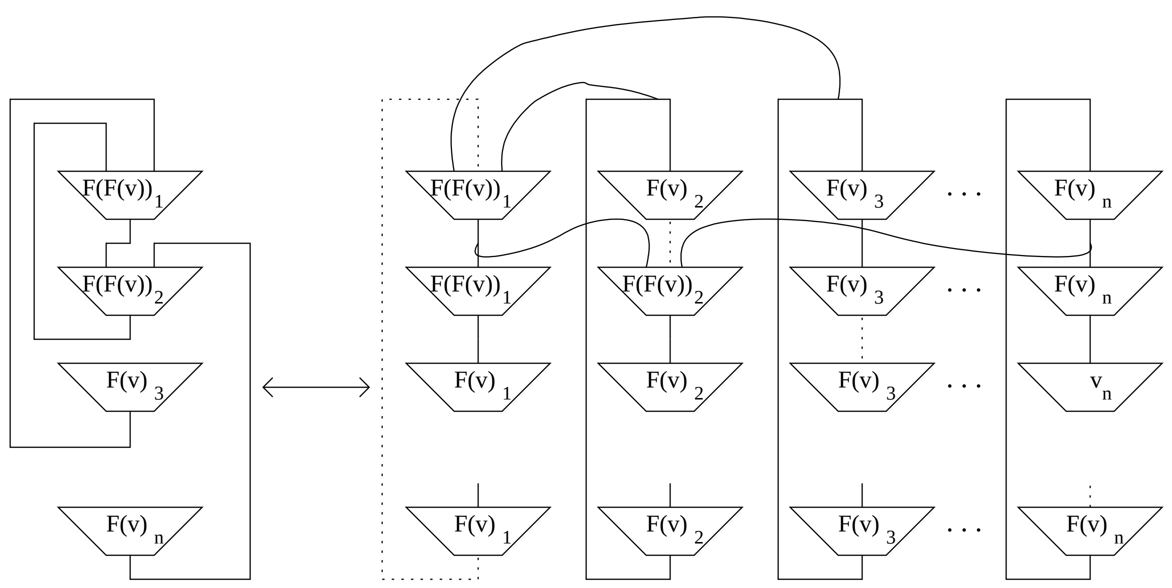

Figure 2: Two steps into updating from $F(\vec{v})$ to $F(F(\vec{v}))$ in the original, gate-by-gate version (left) and in our circuit-wide construction (right). The dashed lines would represent self-loops in the circuit if present, and the labels represent the values output by the gates.

We will frequently discuss *update functions,* $F : [0,1]^{|C|} \to [0,1]^{|C|}$, particularly the following two varieties: *circuit-wide* update functions and *single-gate* update functions. A circuit-wide update function F takes $\vec{x}$ to $F(\vec{x}) = \vec{y}$, where if the ith gate has inputs from the jth and kth gates, then

$$\vec{y}_i = \begin{cases} \max\{x_j, x_k\} & \text{if the } i\text{th gate is a MAX gate} \\ \min\{x_j, x_k\} & \text{if the } i\text{th gate is a MIN gate} \\ (x_j + x_k)/2 & \text{if the } i\text{th gate is an AVG gate} \end{cases}$$

A single-gate update function, F_i is the identity function on each component $j \neq i$, but acts like the circuit-wide update function on the ith component. We will call iterating over the single gate update functions "gate-by-gate update," and will iterate over them in the order of the gate indices. We have introduced these two notions primarily for convenience. In some ways, the circuit-wide update function is more natural, as it does not require an update order to be imposed on the circuit. On the other hand, gate-by-gate update simplifies our constructions, and makes verification of their correctness much easier.

As an aside, one should note that it is easy to convert an n-gate construction that works under gate-by-gate update into an n^2 gate construction that is correct under circuit-wide update: consider the n^2 gates arranged in a square array, with the ith gate along the diagonal of the same type as the ith gate in the original circuit, and the off-diagonal gates of arbitrary types. If gate i in the original circuit had inputs from gates j and k, then gate (i,i) in our new circuit should take inputs from gates $(i-1 \bmod n, j)$ and $(i-1 \bmod n, k)$. Each other gate (i,j) should take both its inputs from gates $(i-1 \bmod n, j)$, so that it acts as a one-step delay. See figure 2 for an example. For any value vector on the original circuit that followed from some round of gate-by-gate update, there is a corresponding value vector on this new circuit, with the ith gate along the diagonal taking the same value as the ith gate in the original circuit, every gate above the ith gate in its column taking the same value as the ith gate had on the previous round, and every gate below the ith gate taking the same value as the ith gate. It is easy to see that after we apply the circuit-wide update function k times, the value vector we would obtain from the kth row of this array is the same as if we had applied single-gate update functions 1–k to the original configuration.

We say that a value vector is *stable* iff every gate is satisfied; equivalently, the stable vectors are the fixed points of our circuit-wide update function F. Since each component of F is clearly a continuous function, F itself is continuous and maps $[0,1]^{|C|}$ into itself, and hence by Brouwer's fixed-point theorem, fixed points (stable vectors) exist. It should be clear that the fixed points of the circuit-wide update function and the gate-by-gate update function are the same. We will be particularly interested in the *minimum stable solution*, $\vec{m}$, which we can define as

$$\vec{m} = \lim_{n\to\infty} (F_{|C|} \circ \cdots \circ F_1)^n(\vec{0})$$

where $\vec{0}$ is the all-zero vector—that is, gate-by-gate update from $\vec{0}$ yields $\vec{m}$ in the limit.

Claim 1. *$\vec{m}$ is well-defined.*

$\vec{m}$ is a minimum in the following strong sense

Lemma 1. *$\forall \vec{s} \in [0,1]^{|C|}$, if $\vec{s}$ is stable, then $\forall k\ m_k \leq s_k$.*

We now define the function problem STABLE-CIRCUIT(C) to be "given a circuit C, find the minimum stable solution $\vec{m}$." We will soon see that it is sufficient to specify answers as either a vector of $4|C|$-bit approximations to the components of $\vec{m}$ or as a mapping from the MIN/MAX gates to LEFT/RIGHT projection gates, projecting the input wire from which the MIN or MAX gates take their output in $\vec{m}$.

2.3 Equivalence of Simple Stochastic Games and MIN/MAX/AVG Circuits

We show that the definitions given in the previous two sections are equivalent in several ways. In particular, suppose we are given a graph G defining a Simple Stochastic Game, and we label the vertices in $V(G) - \{\mathbb{O}, \mathbb{I}\}$ by natural numbers. We will show that such a simple stochastic game defines an equivalent MIN/MAX/AVG circuit. Moreover, we can also show that the decision problem SSG-VALUE is polynomial-time equivalent to the function problem STABLE-CIRCUIT.

2.3.1 Translation Between Circuits and Games

Given a MIN/MAX/AVG circuit C, we can construct the equivalent Simple Stochastic Game, $G(C)$ as follows:

$$\begin{aligned}
MIN &= \{i : \text{gate } i \text{ is a MIN gate}\} \\
MAX &= \{i : \text{gate } i \text{ is a MAX gate}\} \\
AVG &= \{i : \text{gate } i \text{ is a AVG gate}\} \\
E &= \{(i,j) : \text{gate } i \text{ has an input from gate } j\} \\
&\quad \cup\{(i,\mathbb{O}) : \text{gate } i \text{ has a constant input of } 0\} \\
&\quad \cup\{(i,\mathbb{I}) : \text{gate } i \text{ has a constant input of } 1\}
\end{aligned}$$

Likewise, it is clear that given a Simple Stochastic Game G, if we label every $v \in V(G) - \{\mathbb{O}, \mathbb{I}\}$ by the integers $1, 2, \ldots, |V(G) - \{\mathbb{O}, \mathbb{I}\}|$, then we can construct the circuit by creating a gate for each node (where the type of each gate is specified by which of MIN, MAX, or AVG contains the node) and assigning inputs for each edge leaving the node. The circuit we obtain in this way from $G(C)$ is equivalent up to relabeling to the original circuit C. These functions should easily be logspace-computable.

2.3.2 Equivalence of Values and Minimum Stable Solutions

We define $\overrightarrow{value}$ such that if $value(i)$ denotes the value of the simple stochastic game G starting from i, then $value_i = value(i)$. We will also let $\overrightarrow{value}^{\sigma,\tau}$ be the vector such that $value_i^{\sigma,\tau} = value(i, \sigma, \tau)$. We now further introduce "strategies" into our circuits—a strategy σ for the MAX (or MIN) gates of a circuit C is a strategy of the 1-player (or 0-player, respectively) in $G(C)$. We apply the strategy to the circuit C as follows: suppose the ith gate is a MAX gate, and $(i, j) \in \sigma$. Then, we define the single-gate update function under σ, F_i^σ, such that $(F_i^\sigma(v))_i = v_j$. F_i^σ acts as the identity on all other gates, and for any gate k that is not a MAX gate, $F_k^\sigma = F_k$. In this way, we have informally assigned our MAX gates to work as LEFT or RIGHT projection gates according to σ. We can now define gate-by-gate update under σ as $F_{|C|}^\sigma \circ \cdots \circ F_1^\sigma$, and the minimum stable solution under σ, $\vec{m}^\sigma$ as

$$\vec{m}^\sigma = \lim_{n\to\infty} (F_{|C|}^\sigma \circ \cdots \circ F_1^\sigma)^n(\vec{0})$$

just as before. The circuit-wide update function under σ, F^σ, is also defined in the natural way. Similarly, we can define the update functions for a *pair* of strategies, σ and τ for the 1-player and 0-player, respectively. Notice that in this case, we assign every MAX and MIN gate to act as a LEFT or RIGHT projection gate. Consequently, we can write

$$F^{\sigma,\tau}(\vec{v}) = Q\vec{v} + \vec{b}$$

where $Q \in \{0, 1/2, 1\}^{|C|\times|C|}$ and $\vec{b} \in \{0, 1/2, 1\}^{|C|}$. To be precise,

$$Q(i,j) = \begin{cases} 1 & \text{if gate } i \text{ is acting as a projection of gate } j \\ 1/2 & \text{if gate } i \text{ is an AVG gate that takes gate } j \text{ as an input} \\ 0 & \text{otherwise} \end{cases}$$

and

$$b_i = \begin{cases} 1 & \text{if gate } i \text{ acts as a projection of an input of constant 1 or has both inputs of constant 1} \\ 1/2 & \text{if gate } i \text{ is an AVG gate taking one input of constant 1} \\ 0 & \text{otherwise} \end{cases}$$

We will need to consider separately gates in the circuit which take input (directly or indirectly) from hard-wired constants and gates in the circuit which do not.

Claim 2. *For any gate i which is not directly or indirectly connected to a hard-wired input, $m_i = 0$.*

This reduces the difficulty of computing $\vec{m}$ to finding m_i for those gates i which are connected to a hard-wired input. Observe that, if we replace wires from gates in D with hard-wired zero inputs, we do not change $\vec{m}$. It should therefore be clear that WLOG we can consider every gate in the circuit to be connected to a hard-wired input, by making the above modification and then removing the gates in D from the circuit.

Similarly, in the case of a simple stochastic game G,

Claim 3. *For any $i \in V(G)$ which has no path to $\mathbb{O}$ or $\mathbb{I}$, $value_i = 0$.*

We can perform a modification on games that is simliar to what we have done for circuits above, removing those vertices not connected to any sink and replacing edges to them by edges to the 0-sink. Again, it is clear that this does not alter the values of the remaining nodes. Moreover, the reader should note that for any pair of an equivalent circuit and game (as described in the previous section) the circuit and game we obtain by removing nodes not connected to any hard-wired input and vertices with no path to any sink (respectively) are still equivalent.

The following important lemma is due to Condon [5]:

Lemma 2. *Let strategies σ and τ be given, and let Q and $\vec{b}$ be as stated above for the circuit with all gates connected to a hard-wired input.*

Then, $\overrightarrow{value}^{\sigma,\tau}$ is the unique solution to the equation $\overrightarrow{value}^{\sigma,\tau} = Q\overrightarrow{value}^{\sigma,\tau} + \vec{b}$. Also, $I - Q$ is invertible, all entries of $(I-Q)^{-1}$ are non-negative, and the entries along the diagonal are strictly positive.

Together with claim 3, this implies that given a pair of strategies σ and τ, in general $\overrightarrow{value}^{\sigma,\tau}$ is unique, since we know the values of the vertices with no path to a sink to be zero, and we have shown that the rest of the vertices have unique values.

Observe now that we can recover a "strategy" of the MAX gates of the circuit from $\vec{m}$, $\sigma(\vec{m})$, in the natural way—the 1-player in $G(C)$ takes an edge $(i,j) \in \sigma(\vec{m})$ iff for the MAX gate i, $m_i = m_j$ (we can resolve ambiguities by using the first such j for σ), or takes an edge to $\mathbb{O}$ or $\mathbb{I}$ iff the MAX gate i takes its value from a 0 or 1 input, respectively. Clearly, this puts an edge for each MAX vertex in σ, so it is a valid strategy of the 1-player in $G(C)$. We can similarly recover strategies of the MIN player from $\vec{m}$. It should be clear that by lemma 2 and claim 2,, $\vec{m} = \vec{m}^{\sigma(\vec{m}),\tau(\vec{m})}$.

Lemma 3. *Given σ, a strategy of the 1-player for $G(C)$, and a value vector for any strategy of the 0-player (when the 1-player plays according to σ), $\vec{u}$, at each ith gate of C, $m_i^\sigma \leq u_i$.*

Of course, $\tau(\vec{m}^\sigma)$ achieves $\overrightarrow{value}^{\sigma,\tau(\vec{m}^\sigma)} = \vec{m}^\sigma$ by lemma 2 since both are fixed points of $F^{\sigma,\tau(\vec{m}^\sigma)}$. It follows easily that for every ith gate, $m_i^\sigma = \min_\tau value(i,\sigma,\tau)$.

Lemma 4. $\vec{m} = \overrightarrow{value}$

Thus, henceforth we shall discuss simple stochastic games and MIN/MAX/AVG circuits interchangibly.

2.3.3 Polynomial-Time Equivalence of SSG-VALUE and STABLE-CIRCUIT

Now that we have identified $\vec{m}$ with $\overrightarrow{value}$, the reduction from SSG-VALUE(G,i) to STABLE-CIRCUIT is trivial: compute STABLE-CIRCUIT$(C(G))$, and decide "yes" iff $m_i \geq 1/2$. The other direction is a little more interesting. We state another lemma from Condon [5], one that we have implicitly been using for some time:

Lemma 5. *The value of a simple stochastic game G with n non-sink vertices is of the form p/q, where p and q are integers, $0 \leq p, q \leq 4^n$.*

We interpret this as stating that $4|C|$-bit approximations to the entries of $\vec{m}$ are sufficient to recover the precise values of $\vec{m}$. Notice that any two entries, $m_i = p_i/q_i$ and $m_j = p_j/q_j$ which differ must have $|m_i - m_j| \geq 1/q_iq_j > 1/2^{4|C|}$. Hence, we can tell which is greater from the first $4|C|$ bits, allowing us to recover $\sigma(\vec{m})$ and $\tau(\vec{m})$, where by lemma 2, $\vec{m}$ is the unique solution to

$$\vec{m} = F^{\sigma(\vec{m}),\tau(\vec{m})}(\vec{m}) = Q\vec{m} + \vec{b}$$

for an easily computable Q and $\vec{b}$. Thus, by Gaussian elimination, we can efficiently recover the precise values of $\vec{m}$ by solving for $\vec{m}$.

The following simple construction will be useful to us:

Claim 4. *Let $0.b_k \cdots b_1$ be the "binary decimal" representation of a constant in the interval $[0,1)$. We can compute this constant using k AVG gates.*

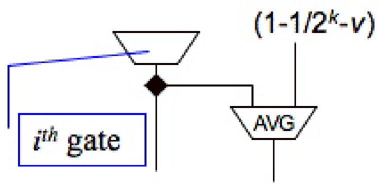

Figure 3: Extracting a bit of m_i

Now, given that we can decide SSG-VALUE(G, i) efficiently, consider the following algorithm for STABLE-CIRCUIT:

Algorithm. On input C,

1. For $i = 1, \ldots, |C|$, repeat step 2:
2. For $k = 1, \ldots, 4|C|$, repeat steps 3 and 4:
3. If $k > 1$, compute C_k, which is identical to C except that there is an AVG gate, j, which takes the ith gate as its first input and the constant $1 - 1/2^k - 0.v_1 \cdots v_{k-1}$ as its second input. If $k = 1$, $C_k = C$, and $j = i$.
4. Put $v_k = 1$ if SSG-VALUE$(G(C_k), j)$ is a "yes" instance, and put $v_k = 0$ otherwise. Output v_k.

Claim 5. *$v_1 \cdots v_{4|C|}$ as output during the ith iteration of step 1 are the first $4|C|$ bits of m_i.*

It is also clear that the algorithm runs in polynomial time, given that our algorithm for SSG-VALUE does. Thus, we see that SSG-VALUE is a decision problem that is polynomial-time equivalent to the function problem STABLE-CIRCUIT.

2.4 Leapfrog Circuits

Observe that a MIN or MAX gate always acts as a projection of one of its two inputs. It is natural to consider whether some particular gate is ever "forced" to take its left (or right) input; we could imagine watching the circuit during its computation (either under circuit-wide or gate-by-gate update), waiting to see if (say) the value of the left input to a MAX gate is above the right input. Such a state could be considered an "accepting state" for the circuit.

To make use of this idea, it will be helpful to think of the right input as a threshold: suppose that for each gate x in the circuit, there is a threshold gate $th(x)$. Let us say that the gate x outputs T iff the value output by x is greater than the value output by $th(x)$. If we were to then feed x as the left input to a MAX gate and $th(x)$ as the right input in the scenario previously described, we would find that the circuit enters an accepting state iff x encodes T. It should be clear that the gate itself serves no purpose except to motivate this model, so henceforth we shall disregard it. We will call circuits in this model "Leapfrog" circuits, for reasons that we will try to elucidate later.

It will be critical for us to allow different gates of the circuit to be considered relative to different thresholds. Suppose $th(C)$ denotes the set of gates that are used as thresholds in our circuit C. We now define for each $i \in th(C)$ the set of gates $\{x \in C : th(x) = i\}$ to be the *ith region* of C.

Leapfrog circuits are an attractive model because they allow our monotone MIN/MAX/AVG circuits to exhibit non-monotone behavior. We will see that under the assumption that $\mathbf{NP} \neq \mathbf{PSPACE}$, that we can compute significantly more using these circuits than we could if we only examined stable configurations of the circuits (i.e., solutions to the stable circuit problem). Inspired by the definition of $\vec{m}$, we define LEAPFROG(C, i, th) to be the following decision problem: "$\exists t$ such that for $\vec{v}^t = (F_{|C|} \circ \cdots \circ F_1)^t(\vec{0})$, $v_i^t > v_{th}^t$?" where F_i is the ith single-gate update function, as before.

2.5 Prior Results

The following important result is again due to Condon [5]:

Theorem 1. SSG − VALUE $\in$ **NP** $\cap$ **coNP**.

The following is also discussed at length in the introductory papers [5] and [2], and briefly mentioned in the SSG Algorithms paper [6]:

Claim 6. *Any instance of a simple stochastic game G can be efficiently transformed into a game G' such that $C(G')$ has a unique stable solution that is arbitrarily close to the minimum stable solution of $C(G)$.*

Papadimitriou [10] has defined a class **PPAD**, containing problems that can be solved efficiently by invoking the lemma, "all graphs contain an even number of odd-degree nodes" in a directed graph. In the same paper, he defines BROUWER, the following computational version of Brouwer fixed-points: "given an integer n and a Turing machine which computes a function $f : [0,1]^d \to [0,1]^d$, with the input given as a vector in coordinate multiples of $1/n$, the output given as a displacement vector of magnitude no greater than $1/n^2$, and the rest of the map implicitly defined by interpolating the values across any cubelet, find an input x such that $f(x) = x$." He is then able to prove the following:

Theorem 2. BROUWER *is* **PPAD**-*complete.*

We will only require the fact that BROUWER $\in$ **PPAD**.

3 Complexity Results for MIN/MAX/AVG Circuits

3.1 Classifying Stable Circuit

We now present some classifications of STABLE-CIRCUIT. Recall that a problem is in **PLS**, introduced in [9], if informally

- An initial feasible solution can easily be computed
- The function under optimization may be efficiently computed for any feasible solution
- Given a locally optimal feasible solution, we can efficiently determine the solution is locally optimal
- Given a feasible solution that is not locally optimal, we can efficiently find a better neighbor

Condon [6] noted that there is a local search algorithm for solving SSGs (e.g., STABLE-CIRCUIT). Since **PLS** was defined with the intent of capturing local search, one would therefore expect the following to hold (which we have confirmed):

Theorem 3. STABLE − CIRCUIT $\in$ **PLS**.

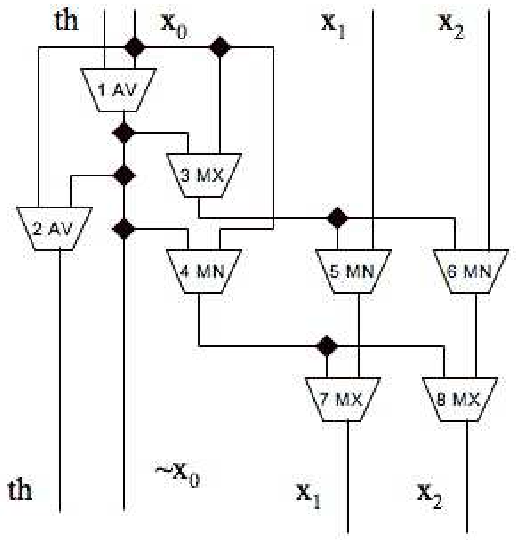

Figure 4: NOT Gadget

This result, while interesting, is much more interesting when one considers the following:

Theorem 4. STABLE − CIRCUIT ∈ **PPAD**.

This makes STABLE-CIRCUIT the first problem outside **P** to be contained in **PLS**∩**PPAD**, to our knowledge.

3.2 PSPACE-hardness of Leapfrog Circuits

In this section, we will build up to a reduction from TQBF to LEAPFROG in several steps, introducing one important gadget or circuit at each step. Before we begin, suppose that the ith region of the circuit, $th^{-1}(i) = \{x_0, \ldots, x_k\}$. Suppose that the gate i outputs some value th_i and $\forall x_j : j = 0, \ldots, k$, $x_j \in \{\mathsf{T}_i, \mathsf{F}_i\}$ where $0 \le \mathsf{F}_i < th_i < \mathsf{T}_i \le 1$—that is to say, in the ith region, there are unique values for truth and falsehood, which are distinct from the threshold value. Observe that, within the ith region now, a MAX gate functions as an OR gate, and a MIN gate functions as an AND gate.

3.2.1 NOT Gadget

We now demonstrate that, by changing regions of the circuit, we can obtain the negation of an arbitrary gate's output. We have taken to calling circuits in this model "Leapfrog" circuits because on this transition, the threshold and the value being negated "leapfrog" over one another. The reader should be aware that, in order to maintain the desired invariants outlined in the introduction to this section, it will be necessary to update every value from the previous region of the circuit when we change regions. We will do this in such a way as to preserve the encodings of each value, except for the one being negated.

Claim 7. *Suppose, in the gadget shown in figure 4, that the wire labeled th entering from above holds* th_i*, the threshold for* $x_0, \ldots, x_k$*, where* $\forall x_j : j = 0, \ldots, k$*,* $x_j \in \{\mathsf{T}_i, \mathsf{F}_i\}$ *such that* $0 \le \mathsf{F}_i < th_i < \mathsf{T}_i \le 1$*. Then, if the gates are updated in the order indicated by their labeling and we take* *the output of the gate labeled 2 to be the threshold th in a new region of the circuit, there will be values $0 \leq \mathsf{F}' < th < \mathsf{T}' \leq 1$ such that for each wire leaving the circuit labeled x_j,*

1. *$x_j \in \{\mathsf{F}', \mathsf{T}'\}$*
2. *The output $\sim x_0 = \mathsf{F}'$ iff the input $x_0 = \mathsf{T}_i$*
3. *$\forall j > 0$ the output $x_j = \mathsf{F}'$ iff the input $x_j = \mathsf{F}_i$.*

We claim that the construction in figure 4 extends in an obvious way to an arbitrary number of wires. Clearly, if there are w wires excluding the wire carrying the threshold, then the gadget has size $2w+2 = \Theta(w)$ gates. The reader is here advised to reconcile the seemingly contradictory set of circumstances—that we have constructed a non-monotone gadget using circuits of monotone gates! Of course, under gate-by-gate update, the sequence of outputs for each gate is still monotone; it is only by interpreting the values relative to one another that non-monotonicity is exhibited.

3.2.2 Boolean Formulas

Now, since we had gates that were equivalent to AND and OR, and now we have a construction for NOT gates, it is easy to build circuits that are equivalent to boolean formulas. It is particularly easy if the formula is given in, say, CNF: suppose we are given n input wires $x_1, \ldots, x_n$ and we wish to evaluate a CNF formula $\phi(x_1, \ldots x_n)$. After n stages of NOT gadgets we can, for each index i, obtain wires carrying x_i and $\sim x_i$. Since each NOT gadget has size $\Theta(n)$, where $|\phi| > n$ this portion has size $O(|\phi|^2)$. Now, for each jth clause C_j, clearly we can use $|C_j| - 1$ MAX gates to compute the OR of the literals in the clause (one new MAX gate for each literal after the first), and to compute the AND of the clauses takes one MIN gate for each clause after the first, so clearly by using only $\Theta(|\phi|)$ MAX and MIN gates, the final MIN gate will output T relative to the threshold iff $\phi(x_1, \ldots, x_n)$ would evaluate to truth on the assignment encoded on the input wires. Hence, it is clear that we can evaluate any formula ϕ using circuits of size $O(|\phi|^2)$, which should be computable from the input formula ϕ in logspace.

3.2.3 Binary Counter

We next show how to construct a more intricate device using the NOT gadgets, one which has n gate outputs in the same region of the circuit that, on the kth round of gate-by-gate update starting from $\vec{0}$, encode the integer $k \pmod{2^n}$ in binary. That is, when used in a circuit in an instance of LEAPFROG, it behaves as a binary counter in this way.

We will denote the least significant bit of the counter by x_0. The construction controlling this bit is particularly easy, since we know that it flips between T and F on every round of gate-by-gate update. Thus, if we pass x_0 to a NOT gadget, we will obtain the value of x_0 for the next round, and may ultimately feed this value back to the first region of the circuit.

The remaining bits of the counter are not so simple. We take a moment now to state that, when we say that there is a "carry out" from the ith bit of the counter, we mean that if we were to increment the value displayed on the counter in the current round, then there would be a carry from the ith position. The reader should pause for a minute to note that the "carry out" value from x_0 is simply the value of the bit from the previous round. Keeping this definition in mind, we consider the circuit diagrammed in figure 5. For simplicty's sake, we have omitted the threshold and any other wires that are "along for the ride" in the region transitions over the NOT gadgets from this diagram. The reader should also note that, when we label the NOT gadget with an integer, we mean that every gate of the NOT gadget should be updated in the order specified when

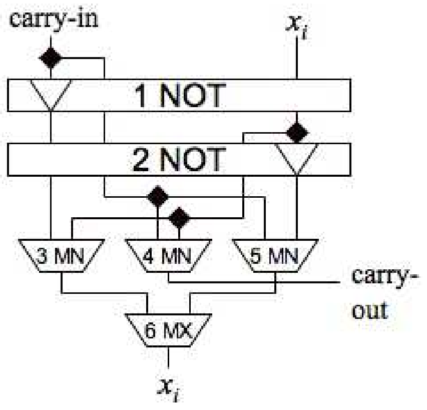

Figure 5: Circuitry for one bit of a binary counter

we constructed the gadget, and these gates should be inserted into the indexing in the position indicated on the gate. When we refer to the "output" of a NOT gadget, we mean the value that was negated during the transition. In the diagram, this is the wire passing through the downward triangle.

Claim 8. *If the incoming wire x_i encodes the ith bit of the counter on the current round and the wire "carry in" encodes whether there would be a carry from the $i-1$st bit into the ith bit if the integer on the counter were incremented, then the outgoing wire x_i encodes the ith bit of the counter on the next round, and the wire "carry out" encodes whether there would be a carry from the ith bit into the $i+1$st bit of the counter.*

We now only need to ensure that our starting encoding has the desired properties, to have a working binary counter gadget. Unfortunately, $\vec{0}$ gives every gate output the same value, so we cannot immediately interpret the encoded truth values relative to thresholds. Suppose now, we include an AVG gate which has one input of constant 1 and its other input from the threshold in the final region of the circuit. If we give this gate the first index, then when it is updated, the gate outputs will be as in the vector $(1/2, 0, 0, \dots)$, so any gate assigned 1 as its threshold will, on the first iteration of the circuit, encode F.

Claim 9. *Suppose the gate i whose output is a threshold has its output passed through an AVG gate in which the second input is a 1. Then, if we take this gate's output to be the threshold th' in a new region of the circuit, for each value $x_j \in \{\mathsf{T}_i, \mathsf{F}_i\}$ in the ith region of the circuit that is also passed through an AVG gate in which the second input is a 1, there will be values $\mathsf{T}' > th' > \mathsf{F}'$ such that the AVG gate outputs T' iff $x_j = \mathsf{T}_i$ and otherwise the AVG gate outputs F'.*

Thus, we may use these AVG gates as a region transition. We choose to give the AVG gate for the threshold gate index 1 and the AVG gates for $x_0, \dots x_{n-1}$ indices $|C| - n + 1, \dots, |C|$. In this way, across rounds of gate-by-gate update, the gates for this region transition are in effect updated consecutively, but the threshold is updated at the beginning of the first step of a round where the rest are updated during the final steps of the previous round. Putting this all together, we can obtain a circuit that acts as a two-bit binary counter, as shown in figure 6. We have collapsed the circuitry controlling x_1 together with the circuitry controlling x_0, since the latter (as discussed earlier) requires no more than a NOT gadget, and we also noted that the previous value

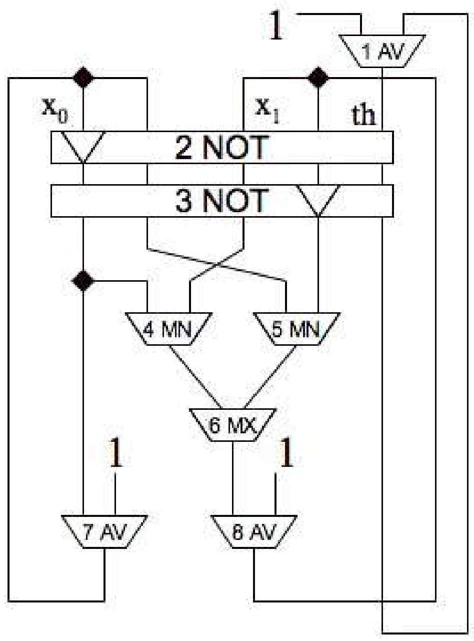

Figure 6: A 2-bit binary counter circuit

of x_0 encoded the "carry out" value. Gates 1, 7, and 8 are as discussed in claim 9, with gate 1 specifically is being used to push the threshold above 0.

It is important to note that, although we have omitted the carry-out wires from this diagram, it is also possible to pass the "carry out" values back to the first region of the circuit. It should be clear that, given n, we could easily output a n-bit counter with these features using only logspace. Notice that, even if we pass each counter bit x_i and its "carry out" value through the entire circuit, this is still $\Theta(n)$ values, so each NOT gadget has size $\Theta(n)$. Since there are two NOT gadgets for each bit of the counter past the first, where clearly the rest of the circuitry for each bit of the counter is constant, each counter bit has size $\Theta(n)$ overall. Hence, the n-bit counter circuit has size $\Theta(n^2)$ overall. As an easy consequence, we find the following:

Theorem 5. SAT $\leq_\ell$ LEAPFROG.

So, we already see that under the assumption that **NP** $\neq$ **coNP**, by Theorem 1, LEAPFROG $\not\leq$ SSG-VALUE. We can and will demonstrate a stronger result next, though.

3.2.4 Universal Quantifier Gadgets

Before we discuss the remaining pieces needed to reduce TQBF to LEAPFROG, we will consider the following interpretation of the construction from Theorem 5 for motivation. Suppose we organize the assignments to $x_0, \ldots, x_{n-1}$ in a binary tree of depth $n+1$ as follows: on a path from the root at the ith level, if we take the left branch we put $x_{i-1} = 0$, and if we take the right branch, we put $x_{i-1} = 1$. If the root of the tree is taken to be the nth level and the level above the leaves is taken to be the 1st level, then our n-bit binary counter is "walking" across the leaves of the tree from left to right; at the ith level of the tree, each time there is a carry out from the $i-2$nd bit, the assignment to x_{i-1} changes, and we switch directions, entering the right subtree of the current

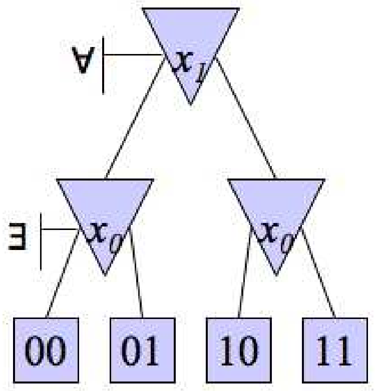

Figure 7: Tree of assignments to two bits

node if there is no carry out from x_{i-1}, and entering an entirely different subtree from a higher level if there is a carry out from x_{i-1}.

We may think of SAT(ϕ) as $\exists x_{n-1} \cdots \exists x_0 \phi(x_0, \ldots, x_{n-1})$, where in this tree, if ϕ evaluates to T under the assignment at any leaf of the tree, we accept. To move to TQBF, we would like to switch one or more of these existential quantifiers to universal quantifiers, yielding a tree such as in figure 7. We will see that this can be done by granting each universally quantified variable one bit of memory (realized by a value being passed through the circuit in a loop) so that it can remember its value on the left branch, adding $\Theta(n)$ gates per quantifier.

Claim 10. *Suppose, in the gadget shown in figure 8, that the input $x_i = \mathsf{F}$ (the current value of the ith bit of the counter), the input "carry-out: x_i" holds the "carry out" value for x_i, that the input A will hold T on some kth round of gate-by-gate update iff the quantified boolean formula $A(x_{n-1}, \ldots, x_{i+1}, \mathsf{F}) = \mathsf{T}$, and that the input v_i^0 holds F when the counter evaluates the assignment $\vec{0}$ and then holds the encoding of the same value output by gate 6 on the previous round. Then, the output of gate 7 is F and the output of gate 6 will hold T after the counter evaluates all settings to $x_0, \ldots, x_{i-1}$ iff $A(x_{n-1}, \ldots, x_{i+1}, \mathsf{F}) = \mathsf{T}$.*

Claim 11. *Suppose, in the gadget shown in figure 8, that the input $x_i = \mathsf{T}$, the input "carry-out: x_i" holds the "carry out" value for x_i, and that the input A is set to T on some round of gate-by-gate update iff the quantified boolean formula $A(x_{n-1}, \ldots, x_{i+1}, \mathsf{T}) = \mathsf{T}$. Also assume that the input v_i^0 holds $A(x_{n-1}, \ldots, x_{i+1}, \mathsf{F})$ Then, the output of gate 7 is set to T during some round of gate-by-gate-update iff $\forall x_i A(x_{n-1}, \ldots, x_{i+1}, x_i) = \mathsf{T}$.*

There are now, for each variable x_i, at most four wires being passed through the circuit—one for the current value of x_i, one for its negation, one for its "carry out" value, and now potentialy one for its left-branch bit, if it is universally quantified. Still, the NOT gadgets have size $\Theta(n)$, since this is just a constant factor. This brings us to the main result:

Theorem 6. TQBF $\leq_\ell$ LEAPFROG

Corollary 1. *If* **NP** $\neq$ **PSPACE***,* LEAPFROG $\not\leq$ SSG – VALUE*.*

Proof. By Theorem 1 SSG-VALUE $\in$ **NP**, and hence LEAPFROG $\leq$ SSG-VALUE would imply **NP** $=$ **PSPACE**, as advertised. ☐

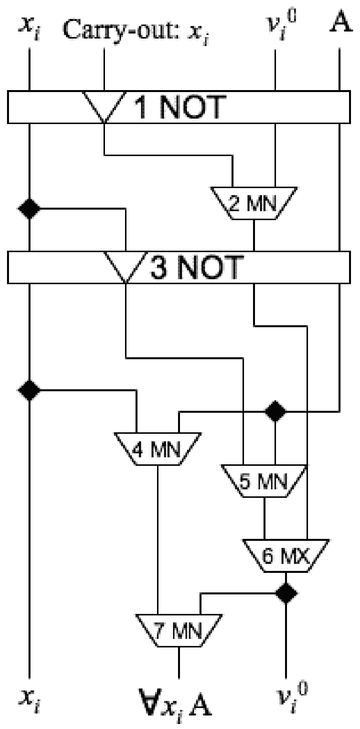

Figure 8: Universal quantifier gadget

3.3 Extending Leapfrog to the Limit

The reader may, at this point, feel that the result for LEAPFROG is rather unremarkable; after all, given that the problem asks for the existence of some index t such that the value output by the ith gate sneaks above the value output by the gate th. It might seem that the hardness of LEAPFROG is due only to the complicated pattern of values rising and falling with respect to their thresholds, which is clearly irrelevant to the limiting values. While it is true that we have shown that the pattern of rising and falling is irrelevant to the limiting values, it turns out that for not much more effort, we can obtain a result that suggests more about the relationship between Leapfrog circuits and their minimum stable solutions.

We define LEAPFROG-LIMIT(C, i, th) to be the decision problem "$\exists N$ such that $\forall n > N$, if $\vec{v}^n = (F_{|C|} \circ \cdots \circ F_1)^n(\vec{0})$, $v_i^n > v_{th}^n$?" where F_i is the ith single-gate update function, as before. The reader should take note that if LEAPFROG-LIMIT(C, i, th) is a "yes" instance, then in the minimum stable solution, $\vec{m}$, $m_i \geq m_{th}$. Likewise, if it is a "no," then $m_i \leq m_{th}$. Moreover, if $m_i > m_{th}$, then LEAPFROG-LIMIT(C, i, th) is a "yes" instance, and if $m_i < m_{th}$, then clearly LEAPFROG-LIMIT(C, i, th) is a "no" instance—that is, the only ambiguity occurs when $m_i = m_{th}$. Keeping these considerations in mind, observe the following (see the Appendix for proof):

Theorem 7. TQBF $\leq_\ell$ LEAPFROG − LIMIT

Corollary 2. *If* **NP** $\neq$ **PSPACE***,* LEAPFROG − LIMIT $\not\leq$ SSG − VALUE.

Proof. LEAPFROG-LIMIT is **PSPACE**-hard, so if LEAPFROG-LIMIT $\leq$ SSG-VALUE, we would find **NP** = **PSPACE** by Theorem 1. □

Thus, we see that under the assumption $\mathbf{NP} \neq \mathbf{PSPACE}$, it is *necessarily* the case that, in some Leapfrog circuits, $m_i = m_{th}$, e.g., that the encodings of T and F must approach each other, and this collapse of the encodings cannot be halted by means of some clever gadget.

4 Discussion and Open Problems

The primary motivation for this work was the wealth of work that has been put into finding a polynomial-time algorithm for SSG-VALUE, and the apparent lack of successes [6]. We felt that the problem was ripe for a hardness result, and set about trying to devise gadgets to this end. Leapfrog circuits seemed to be a natural model to build gadgets under, and particularly after discovering how to build a NOT gadget, we tried to combat the growth of the bits required by the encodings.

To get a sense of this, the reader is advised to quickly trace through the 2-bit binary counter in figure 6. Observe that each time the values pass through a NOT gadget, the AVG gates cause the number of bits of precision required to represent the values to increase by two. The AVG gates used in the transition from the final region of the circuit back to the first region of the circuit then add another required bit of precision *per iteration*. Hence, observe that the number of bits of precision needed to represent the values in a distinguishable way as increasing by five in each round. It should be clear that, since an exponential number of rounds is required in general for the counter to have stepped through all configurations and the number of bits by which the required precision grows is proportional to the number of NOT gadgets – a linear number of bits *per round* – the number of bits per value required by the 2^nth round is $\Theta(n2^n)$.

We had tried to fight this growth in hopes of obtaining a reduction to SSG-VALUE, which would provide us with our desired hardness result. Once we realized what these NOT gadgets permitted us to do (e.g., Theorem 6), it became evident that we would be unlikely to keep the encodings distinguishable from some finite bit position. Theorem 7 and the related discussion in Section 3.3 frame this difficulty in a much more precise way. Consequently, although Leapfrog circuits are quite an interesting model in their own right, their relationship to the stable configurations of the circuits (i.e. and therefore to SSG-VALUE) is more distant than we had hoped.

Returning to STABLE-CIRCUIT, the oracle separations of **PLS** and **PPAD** by Morioka and Buresh-Oppenheim [3] raise the possiblity that the classes are distinct. Clearly, a hardness result for STABLE-CIRCUIT relative to either of these classes would provide us with a containment one way or the other, but given that absolute separations between **P** and **NP** are (to put it mildly) quite rare, it seems possible that we may be left wondering where exactly the problem lies indefinitely.

4.1 Open Problems

Of course, we would still like to know where STABLE-CIRCUIT lies. In particular, if it is hard for either of **PLS** or **PPAD**, this would have the added value of demonstrating a containment between these two classes. Barring this, we would like to know if it is interreducible with some other problem in **PLS** or **PPAD** that has not been shown to be complete for either class. Until recently, the Nash Equilibrium problem for two-player normal form games was such a candiate, and this approach was originally suggested to us by Schoenebeck [11]. A recent breakthrough by Chen and Deng [4], building on recent work by Daskalakis et al. [7] has shown that this problem is, contrary to expectations, **PPAD**-complete, but in spite of this, the problem remains a strong candidate for reduction to STABLE-CIRCUIT. It is also worth noting that the techniques introduced by Daskalakis et al., notably the "averaging maneuver" used to mitigate the problems caused by garbage values produced by their "brittle comparitor" are likely to be inspiring to anyone working with a real-valued network, if not directly applicable.

Returning to LEAPFROG now, for a moment, recall that we observed that the number of bits required to represent the encodings in a distinguishable way throughout the sequence of all 2^n configurations of an n-bit binary counter was exponential in n. Although we stopped after showing the **PSPACE**-hardness of LEAPFROG since this was the most powerful result we could show under our AND/OR/NOT interpretation (after all, AND/OR/NOT circuits can be evaluated in **PSPACE**), it is not clear that this is the most powerful result possible. That the circuits we used repeat configurations after 2^n steps is not sufficient to conclude that this is so in general. In fact, it is clear that the time and space needed to simulate the Leapfrog circuits "into the limit" in the straightforward way are not bounded. We wonder whether or not there is some more clever way to evaluate the circuits, giving us some sort of bound on the resources required.

It is clear that LEAPFROG $\in$ **RE**, since we can always perform the simulation until the finite index where the ith gate's output rises above its threshold's output, but this is quite a weak statement. It is currently unclear whether or not Leapfrog circuits can be made to function as a UTM, which would naturally imply that **RE** is as good as we can do. It would be interesting to see if this is the case, or if we can place LEAPFROG in some more modest class, such as **EXP**.

Acknowledgements

The authors would like to thank Anupam Gupta, Christos Papadimitriou, and Grant Schoenebeck for their contributions.

References

[1] Avrim Blum. Communication.

[2] Manuel Blum, Rachel Rue, and Ke Yang. On the complexity of MAX/MIN/AVRG circuits. Technical Report CMU-CS-02-110, Department of Computer Science, Carnegie Mellon University, 2002.

[3] Joshua Buresh-Oppenheim and Tsuyoshi Morioka. Relativized NP search problems and propositional proof systems. Technical Report TR03-084, Electronic Colloquium on Computational Complexity, 2003.

[4] Xi Chen and Xiaotie Deng. Settling the complexity of 2-player nash-equilibrium. Technical Report TR05-140, Electronic Colloquium on Computational Complexity, 2005.

[5] Anne Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.

[6] Anne Condon. On algorithms for simple stochastic games. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 13:51–73, 1993.

[7] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. Technical Report TR05-115, Electronic Colloquium on Computational Complexity, 2005.

[8] Alan J. Hoffman and Richard Karp. On nonterminating stochastic games. *Management Science*, 12:359–370, 1966.

[9] David S. Johnson, Christos H. Papadimtriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.

[10] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

[11] Grant Schoenebeck. Communication.

[12] Lloyd S. Shapley. Stochastic games. *Proc. National Academy of Sciences*, 39:1095–1100, 1953.

Appendix

The reader is advised that lemma 2, lemma 5, and theorem 1 are from [5], and that theorem 2 is from [10]. Claim 6 is developed implicitly in [5], and its uses are discussed briefly in [6] and [2]. As such, we have not included the proof of these claims, and the interested reader is referred to the cited articles.

Claim (1). *$\vec{m}$ is well-defined.*

Proof. Consider the sequence of vectors, $\vec{v}^i = (F_{|C|} \circ \cdots \circ F_1)^i(\vec{0})$. Since $\forall i, k\ v_k^i \in [0,1]$, it follows by induction that $\forall k \forall i > j,\ v_k^i \geq v_k^j$. Then, $\forall i, k$, since $v_k^i \leq 1$ and $\{v_k^i\}_{i=1}^{\infty}$ is monotone, it has a limit. Thus, $\{\vec{v}^i\}_{i=1}^{\infty}$ has a limit as well. □

Lemma (1). *$\forall \vec{s} \in [0,1]^{|C|}$, if $\vec{s}$ is stable, then $\forall k\ m_k \leq s_k$.*

Proof. Because $\vec{s}$ is stable, we have $\forall k\ F_k(\vec{s}) = \vec{s}$. We will again consider the sequence of vectors $\{\vec{v}^i\}_{i=1}^{\infty}$ from the proof of claim 1. It follows by induction and the stability of $\vec{s}$ that $\forall i, k\ v_k^i \leq s_k$. Consequently,

$$\forall k\ m_k = \lim_{i \to \infty} v_k^i \leq s_k$$

□

Claim (2). *For any gate i which is not directly or indirectly connected to a hard-wired input, $m_i = 0$.*

Proof. First, consider the set $D = \{i : \text{gate } i \text{ is not connected to a hard-wiredinput}\}$. Observe that $\forall i \in D$, if i takes its inputs from gates j and k, then if $j \notin D$ or $k \notin D$, i would be connected to a hard-wired input, which would be a contradiction. Observe that since $\vec{m}$ is obtained by gate-by-gate update from $\vec{0}$, initially every gate in D outputs zero.

Now, let i be the gate in D whose output first rises above zero under gate-by-gate update. Since the inputs to i, j and k, are also in D, theymust still update zero on the application of F_i which sets the output of i above zero. But now regardless of the type of gate, i will still output zero after the application of F_i, a contradiction. □

Claim (3). *For any $i \in V(G)$ which has no path to $\mathbb{O}$ or $\mathbb{I}$, $value_i = 0$.*

Proof. Observe that if we consider the set of edges chosen during a simple stochastic game, these edges form a connected path. Thus, given the token is at vertex i,there must exist a path from i to $\mathbb{I}$ for the token to reach $\mathbb{I}$. Since no such path exists, there are no outcomes for the game in which thetoken reaches $\mathbb{I}$, and thus by definition, $value_i = 0$. □

Lemma (3). *Given σ, a strategy of the 1-player for $G(C)$, and a value vector for any strategy of the 0-player (when the 1-player plays according to σ), $\vec{u}$, at each ith gate of C, $m_i^\sigma \le u_i$.*

Proof. First, observe that in every ith component, $0 \le u_i$. It follows easily by induction that for all n, $((F_{|C|}^\sigma \circ \cdots \circ F_1^\sigma)^n(\vec{0}))_i \le u_i$: we assumed that the 1-player played by σ in $\vec{u}$ and the value of an AVG vertex in $G(C)$ must be the average of the values at the vertices it has edges to in any valid value vector. Thus, the only values which might be altered by any F_j^σ might be at MIN gates, which could only decrease from u_j. Therefore, now, we see that

$$m_i^\sigma = (\lim_{n\to\infty}(F_{|C|}^\sigma \circ \cdots \circ F_1^\sigma)^n(\vec{0}))_i \le u_i$$

as promised. □

Lemma (4). $\vec{m} = \overrightarrow{value}$

Proof. Let any σ, a strategy of the 1-player, be given. Now, since in every ith component, $0 \le m_i$, we find by induction on n that for every ith component, $((F_{|C|}^\sigma \circ \cdots \circ F_1^\sigma)^n(\vec{0}))_i \le m_i$ since σ could do no better than to select the maximum at each MAX gate, and otherwise $F_j^\sigma(\vec{m}) = m_j$. Thus, $m_i^\sigma \le m_i$. Of course, again, if the 1-player uses $\sigma(\vec{m})$, then $\vec{m} = \vec{m}^{\sigma(\vec{m}),\tau(\vec{m})}$ by lemma 2 and claim 2.

Now, for each i, consider σ^* such that $m_i^{\sigma^*} = \max_\sigma m_i^\sigma$. Since we have argued that $m_i^{\sigma^*} \le m_i$, it should be clearthat $\sigma(\vec{m})$ achieves the same value as σ^*, and hence $m_i = \max_\sigma m_i^\sigma$. Moreover, any $m_i^\sigma = \min_\tau m_i^{\sigma,\tau}$ by lemma 3. Thus, by lemma 3 and our definition of $\overrightarrow{value}$, we further find

$$m_i = \max_\sigma \min_\tau value_i^{\sigma,\tau} = value_i$$

□

Claim (4). *Let $0.b_k \cdots b_1$ be the "binary decimal" representation of a constant in the interval $[0,1)$. We can compute this constant using k AVG gates.*

Proof. The constant $0.b_1$ is easy to compute using a single gate: use 0 and $b_1 \in \{0,1\}$ as the inputs to an AVG gate. Inductively suppose we have computed $0.b_i \cdots b_1$ using i AVG gates. Now, suppose we pass the output of the AVG gate that computes this value as the first input to an AVG gate that has b_{i+1} as its second input. Clearly, the output is $0.b_{i+1}b_i \cdots b_1$, and we have only used $i+1$ AVG gates, so the claim holds. □

Claim (5). *$v_1 \cdots v_{4|C|}$ as output during the ith iteration of step 1 are the first $4|C|$ bits of m_i.*

Proof. Suppose we consider the binary decimal representation of $m_i = 0.b_1b_2\cdots$. When $k=1$ in the ith iteration of step 1, clearly $v_1 = b_1$, since SSG-VALUE(G,i) is a "yes" instance iff $b_1 = 1$. Now, suppose inductively that at the kth iteration of step 2, $\forall j < k$, $v_j = b_j$. Then,

$$\begin{aligned}\frac{1}{2}(1 - 1/2^k - 0.v_1\cdots v_{k-1} + 0.b_1b_2\cdots b_{k-1}b_k\cdots) &= \frac{1}{2}(0.\overbrace{1\cdots 1}^{k} + 0.\overbrace{0\cdots 0}^{k-1}b_k\cdots) \\ &= \frac{1}{2}(b_k.\cdots) \\ &= 0.b_k\cdots\end{aligned}$$

so clearly the output of the AVG gate in C_k is greater than or equal to 1/2 iff $b_k = 1$, and hence $v_k = b_k$. □

Theorem (3). STABLE − CIRCUIT $\in$ **PLS**.

Proof. (Sketch) Briefly, this is so because we can use one step of the Hoffman-Karp algorithm for strategy improvement (see [8] or [6]) to define a unique neighbor to each value vector. Since one proof of correctness (as in [6]) of the algorithm involves demonstrating that no component of the value vector decreases and at least one component increases after each iteration, the L_1 norm (for example) suffices as a potential function. □

Theorem (4). STABLE − CIRCUIT $\in$ **PPAD**.

Proof. (Sketch) We will reduce STABLE-CIRCUIT to BROUWER: Given a circuit C we first construct the circuit C' as described in claim 6, which has a unique stable solution that is within $4|C|$ bits of the minimum stable solution of C, and let F' be the circuit-wide update function for C'. Define $g(x) = \left(1 - \frac{1}{|C'|2^{8|C|}}\right)x + \frac{1}{|C'|2^{8|C|}}F'(x)$. Now, clearly $|x - F'(x)| \leq |C'|$, so $|x - g(x)| \leq 2^{-8|C|}$. Hence, if we put $n = 2^{4|C|}$, our displacement vectors $\mu(x)$ have length $|\mu(x)| \leq 1/n^2$, satisfying the conditions for BROUWER, where we can easily construct a Turing machine to compute these displacements. Now, since g and F' have the same fixed points, where the unique fixed point of F' is within $4|C|$ bits of the solution to STABLE-CIRCUIT(C), the first $4|C|$ bits of the fixed point to g is our solution. Hence, by Theorem 2, STABLE-CIRCUIT$\in$ **PPAD**. □

Claim (7). *Suppose, in the gadget shown in figure 4, that the wire labeled th entering from above holds* th_i*, the threshold for* $x_0, \ldots, x_k$*, where* $\forall x_j : j = 0, \ldots, k$*,* $x_j \in \{\mathsf{T}_i, \mathsf{F}_i\}$ *such that* $0 \leq \mathsf{F}_i < th_i < \mathsf{T}_i \leq 1$*. Then, if the gates are updated in the order indicated by their labeling and we take the output of the gate labeled 2 to be the threshold th in a new region of the circuit, there will be values* $0 \leq \mathsf{F}' < th < \mathsf{T}' \leq 1$ *such that for each wire leaving the circuit labeled* x_j*,*

1. $x_j \in \{\mathsf{F}', \mathsf{T}'\}$
2. *The output* $\sim x_0 = \mathsf{F}'$ *iff the input* $x_0 = \mathsf{T}_i$
3. $\forall j > 0$ *the output* $x_j = \mathsf{F}'$ *iff the input* $x_j = \mathsf{F}_i$.

Proof. We observe that the gates labeled 1, 2, 3, and 4 are updated first. Let th be the output of gate 2, T' be the output of gate 3, and F' be the output of gate 4. Notice that the output of gate 2 is

$$\frac{1}{2}\left(x_0 + \frac{1}{2}(x_0 + th_i)\right) = \frac{3}{4}x_0 + \frac{1}{4}th_i$$

While the output of gate 3 is $\max\left\{\frac{1}{2}(x_0 + th_i), x_0\right\}$ and the output of gate 4 is $\min\left\{\frac{1}{2}(x_0 + th_i), x_0\right\}$.

Case i. $th_i < x_0$. Clearly,

$$0 \leq \mathsf{F}_i < \mathsf{F}' < th < \mathsf{T}' = \mathsf{T}_i \leq 1$$

Now, $x_0 = \mathsf{T}_i$, and we see $\sim x_0 = \mathsf{F}'$. Moreover, for any x_j with $j > 0$, if the input $x_j = \mathsf{T}_i$, then clearly the output $x_j = \mathsf{T}_i = \mathsf{T}'$. Likewise, if the input $x_j = \mathsf{F}_i$, then in the MAX gates (e.g. gates 7, 8, etc. in figure 4) the output $x_j = \max\{\mathsf{F}', \mathsf{F}_i\} = \mathsf{F}'$.

Case ii. $th_i > x_0$ holds, so

$$0 \leq \mathsf{F}_i = \mathsf{F}' < th < \mathsf{T}' < \mathsf{T}_i \leq 1$$

This time, $x_0 = \mathsf{F}_i$ and $\sim x_0 = \mathsf{T}'$. Now, given any x_j with $j > 0$, if the input $x_j = \mathsf{T}_i$, then in the MIN gates (gates 5, 6, etc.) we find that the output is $\min\{\mathsf{T}', \mathsf{T}_i\} = \mathsf{T}'$, and then the output labled x_j will be T' as well. If, on the other hand, the input $x_j = \mathsf{F}_i = \mathsf{F}'$, then clearly the output labeled x_j will be F' as well.

□

Claim (8). *If the incoming wire x_i encodes the ith bit of the counter on the current round and the wire "carry in" encodes whether there would be a carry from the $i-1$st bit into the ith bit if the integer on the counter were incremented, then the outgoing wire x_i encodes the ith bit of the counter on the next round, and the wire "carry out" encodes whether there would be a carry from the ith bit into the $i+1$st bit of the counter.*

Proof. It will be helpful to recall that we assumed in our encoding that $\mathsf{T} > \mathsf{F}$.

Case i. carry-in encodes T, x_i encodes T.

Then, since the output of NOT 1 is F, the output of MIN 3 is F, and since the output of NOT 2 is F, the output of MIN 5 is F. Hence, the output of MAX 6 is F, as x_i should be on the next round. Likewise, it is clear that there would have been a carry out of the ith bit, and since both carry-in and x_i encode T, MIN 4 will output T, which is the "carry out" value desired.

Case ii. carry-in encodes T, x_i encodes F.

Here, since carry-in encodes T and the output of NOT 2 will encode T, MIN 5 will output T, and hence MAX 6 will output T, which should be the value of x_i on the next round. Also, since x_i was F, notice MIN 4 will output F, where there would not have been a carry out.

Case iii. carry-in encodes F, x_i encodes T.

This time, x_i encodes T and the output of NOT 1 will encode T, so the output of MIN 3 will be T, and hence again MAX 6 must output T, as it should. Since carry-out encoded F, MIN 4 outputs F, and this is consistent with the fact that there is again no carry out.

Case iv. carry-in encodes F, x_i encodes F.

Again, since neither encodes T, clearly MAX 4 will output F, where this is correctly indicating that no carry will occur out of the ith bit. Since x_i encodes F, MIN 3 will output F, and since carry-in encodes F, MIN 5 also outputs F, and hence MAX 6 must output F as well, which is the value of x_i on the next round.

□

Claim (9). *Suppose the gate i whose output is a threshold has its output passed through an AVG gate in which the second input is a 1. Then, if we take this gate's output to be the threshold th' in a new region of the circuit, for each value $x_j \in \{\mathsf{T}_i, \mathsf{F}_i\}$ in the ith region of the circuit that is also passed through an AVG gate in which the second input is a 1, there will be values $\mathsf{T}' > th' > \mathsf{F}'$ such that the AVG gate outputs T' iff $x_j = \mathsf{T}_i$ and otherwise the AVG gate outputs F'.*

Proof. Recall again that $\mathsf{T}_i > th_i > \mathsf{F}_i$. Put $th' = (th_i + 1)/2$, $\mathsf{T}' = (\mathsf{T}_i + 1)/2$, and $\mathsf{F}' = (\mathsf{F}_i + 1)/2$. Now, $\mathsf{T}' > th' > \mathsf{F}'$. Moreover, since $x_j \in \{\mathsf{T}_i, \mathsf{F}_i\}$, if $x_j = \mathsf{T}_i$, then AVG$(x_j, 1) = \mathsf{T}'$ and otherwise since $x_j = \mathsf{F}_i$, then AVG$(x_j, 1) = \mathsf{F}'$. □

Theorem (5). SAT $\leq_\ell$ LEAPFROG.

Proof. On input ϕ, we observed in Section 3.2.2 that we could output a circuit that computes $\phi(x_1, \dots, x_n)$ using $O(|\phi|^2)$ gates. It should be clear that we can output an n-bit binary counter using $\Theta(n^2)$ gates, with the $O(|\phi|^2)$ size formula circuits taking their inputs from the first region of the binary counter (e.g. the gate outputs encoding the binary integers), for a circuit of size $O(|\phi|^2)$ overall. Our LEAPFROG instance is this circuit, examining the output of the formula circuit relative to its threshold.

Since the counter places the binary encoding of $k \pmod{2^n}$ on the inputs to the formula circuits during the kth round of gate-by-gate update, if the formula has some satisfying assignment, then on the round such that the satisfying assignment interpreted as a binary integer is the round number, the output of the formula circuit will be strictly above its threshold, so the LEAPFROG instance will be a "yes." On the other hand, if no such assignment exists, then on every round, the output of the formula circuit will be strictly below threshold, so the LEAPFROG instance will be a "no." Thus, since the computation of these polynomial-sized circuits could clearly be carried out in logspace, we have logspace-reduced SAT to LEAPFROG. $\square$

Claim (10). *Suppose, in the gadget shown in figure 8, that the input $x_i = \mathsf{F}$ (the current value of the ith bit of the counter), the input "carry-out: x_i" holds the "carry out" value for x_i, that the input A will hold T on some kth round of gate-by-gate update iff the quantified boolean formula $A(x_{n-1}, \dots, x_{i+1}, \mathsf{F}) = \mathsf{T}$, and that the input v_i^0 holds F when the counter evaluates the assignment $\vec{0}$ and then holds the encoding of the same value output by gate 6 on the previous round. Then, the output of gate 7 is F and the output of gate 6 will hold T after the counter evaluates all settings to $x_0, \dots, x_{i-1}$ iff $A(x_{n-1}, \dots, x_{i+1}, \mathsf{F}) = \mathsf{T}$.*

Proof. First observe that, when the counter enters the subtree rooted at the current assignment to $x_{n-1}, \dots, x_i$, since $x_i = \mathsf{F}$, if we are not at the assignment $\vec{0}$, then there was a carry out from x_i. Hence, in this case the ouput of NOT 1 will be F, and under the assignment $\vec{0}$ we assumed the input $v_i^0 = \mathsf{F}$, so in either case we find that the output of MIN 2 is initially F.

Now, notice that since $x_i = \mathsf{F}$, the output of NOT 3 is T. Hence, if $A(x_{n-1}, \dots, x_{i+1}, \mathsf{F}) = \mathsf{T}$ on some kth round, then clearly the output of gate 5 will be T, so the output of gate 6 is T. Likewise, if on some previous iteration while $x_i = \mathsf{F}$, gate 6 output T, then since by assumption, v_i^0 is passed back through the circuit and there will not be another carry out of x_i until $x_i = \mathsf{T}$, the output of gate 2 will be T, and so the output to gate 6 will continue to be T.

On the other hand, if $A(x_{n-1}, \dots, x_{i+1}, \mathsf{F}) = \mathsf{F}$, then the output to gate 5 will always be F, and since we saw earlier that the output of gate 2 is initially F, the output to gate 6 will be F. This will hold on the following iterations, as we pass this value for v_i^0 back around, since the output to gate 2 is again F.

To see that the output to gate 7 stays F, we need only observe that $x_i = \mathsf{F}$, so the output of gate 4 will be F, and hence gate 7 will output F as well. $\square$

Claim (11). *Suppose, in the gadget shown in figure 8, that the input $x_i = \mathsf{T}$, the input "carry-out: x_i" holds the "carry out" value for x_i, and that the input A is set to T on some round of gate-by-gate update iff the quantified boolean formula $A(x_{n-1}, \dots, x_{i+1}, \mathsf{T}) = \mathsf{T}$. Also assume that the input v_i^0 holds $A(x_{n-1}, \dots, x_{i+1}, \mathsf{F})$ Then, the output of gate 7 is set to T during some round of gate-by-gate-update iff $\forall x_i A(x_{n-1}, \dots, x_{i+1}, x_i) = \mathsf{T}$.*

Proof. Notice that since $x_i = \mathsf{T}$, the output of NOT 3 is F, so the output of gate 5 is F, and hence the output of gate 6 is v_i^0, Since, by assumption, v_i^0 holds the evaluation of $A(x_{n-1}, \dots, x_{i+1}, \mathsf{F})$, if this is F, then clearly $\forall x_i A(x_{n-1}, \dots, x_{i+1}, x_i)$ evaluates to F as well—where since the output of gate 6 is then F, the output of gate 7 will always be F, as needed.

Now suppose instead $v_i^0 = \mathsf{T}$. If $A(x_{n-1}, \dots, x_{i+1}, \mathsf{T}) = \mathsf{T}$, then by assumption on some round during evaluation of $x_0, \dots, x_{i-1}$, the input A will be T, and since $x_i = \mathsf{T}$ as well, the output to gate 4 will be T. Moreover, since we assumed $v_i^0 = \mathsf{T}$, this means $A(x_{n-1}, \dots, x_{i+1}, \mathsf{F})$ should evaluate to T, and consequently $\forall x_i A(x_{n-1}, \dots, x_{i+1}, x_i)$ should also evaluate to T. Of course now, both inputs to gate 7 are T, so gate 7 outputs T on this round, as needed.

On the other hand, if $A(x_{n-1}, \dots, x_{i+1}, \mathsf{T}) = \mathsf{F}$ then by hypothesis gate 4 will always output F, and therefore gate 7 will output F as well. Naturally, since $\forall x_i A(x_{n-1}, \dots, x_{i+1}, x_i)$ should evaluate to F in this case, this is correct. □

Theorem (6). TQBF $\leq_\ell$ LEAPFROG

Proof. Suppose our instance is of the form $Q_{n-1}x_{n-1} \cdots Q_0 x_0 \phi(x_{n-1}, \dots, x_0)$. We then build the circuit as in Theorem 5 for $\phi(x_{n-1}, \dots, x_0)$, additionally passing the "carry out" values and an extra value v_i^0 for each universally quantified x_i through the counter. As observed above, this only increases the number of wires being passed between regions of the circuit by a constant factor, so the size of the circuit thus far is still $O(|\phi|^2)$.

Now, for each $Q_i = \forall$, we chain a quantifier gadget onto the output of the formula circuit for $\phi(x_{n-1}, \dots, x_0)$—we pass the evaluation of this formula to the first quantifier gadget, and pass the output of gate 7 from the previous quantifier gadget as the wire marked A in the next. Each quantifier $Q_i = \forall$ is passed the current value from the counter of x_i, the "carry out" value for x_i, and the value v_i^0 in the other labeled inputs with its outputs labeled x_i and v_i^0, of course, taken to be the current values for these wires subsequently. As usual, we pass $\Theta(n)$ values between regions of the circuit through the two NOT gadgets used in each quantifier gadget where there are clearly $O(n)$ quantifiers, so this chain of quantifier gadgets takes $O(n^2)$ gates, leaving our total for the circuit at $O(|\phi|^2)$. We take our LEAPFROG instance to be this circuit, asking about the output of the copy of gate 7 in the final quantifier gadget, relative to its threshold. As usual, this construction can clearly be done in logspace.

Assume inductively that the jth quantifier gadget (on x_i) will output T iff

$$\forall x_i Q_{i-1} x_{i-1} \cdots Q_0 x_0 \phi(x_{n-1}, \dots, x_0)$$

is true under the assignment specified by $x_{n-1}, \dots, x_{i+1}$ on the counter while evaluating all settings of the lower $i+1$ bits—for the base case, we simply observe that the unquantified formula is evaluated by our formula gadget.

Now, suppose the $j+1$st quantifier gadget is on x_k. Then, since during evaluation of the first k bits, the counter will evaluate

$$\forall x_i Q_{i-1} x_{i-1} \cdots Q_0 x_0 \phi(x_{n-1}, \dots, x_0)$$

on all settings of $x_{k-1}, \dots, x_{i+1}$, we see that the input A for the $j+1$st quantifier gadget will be set to T iff under the current assignment to $x_{n-1}, \dots, x_k$,

$$\exists x_{k-1} \cdots \exists x_{i+1} \forall x_i Q_{i-1} x_{i-1} \cdots Q_0 x_0 \phi(x_{n-1}, \dots, x_0) = \mathsf{T}$$

where, since x_k is the next universally quantified variable, this formula is clearly

$$Q_{k-1} x_{k-1} \cdots Q_0 x_0 \phi(x_{n-1}, \dots, x_0).$$

Now, by claim 10, we see that v_k^0 will be set to T while $x_k = \mathsf{F}$ and the counter evaluates all settings of the lower k bits iff

$$Q_{k-1} x_{k-1} \cdots Q_0 x_0 \phi(x_{n-1}, \dots, x_{k+1}, \mathsf{F}, x_{k-1}, \dots, x_0)$$

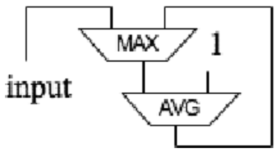

Figure 9: Register circuit

evaluates to T, and hence by claim 11, when x_k is set to T, the output of the $j+1$st quantifier gadget will be set to T at some point while the counter evaluates all settings of the lower k bits iff

$$\forall x_k Q_{k-1} x_{k-1} \cdots Q_0 x_0 \phi(x_{n-1}, \ldots, x_0)$$

evaluates to T under the current, fixed setting to $x_{n-1}, \ldots, x_{k+1}$.

Hence, the quantifier gadgets evaluate the formulas as promised, up to the final universal quantifier. Now we only observe that, if the final universally quantified variable is x_k, then since the counter tries all settings to the existentially quantified $x_{n-1}, \ldots, x_{k+1}$, the final quantifier gadget outputs T and is hence a "yes" instance of LEAPFROG iff the formula $Q_{n-1}x_{n-1} \cdots Q_0 x_0 \phi(x_{n-1}, \ldots, x_0)$ is in TQBF. Since otherwise the final quantifier gadget always outputs F, making this a "no" instance of LEAPFROG, we see that we have sucessfully logspace-reduced TQBF to LEAPFROG. □

Theorem (7). TQBF $\leq_\ell$ LEAPFROG – LIMIT

Proof. Given ϕ, an instance of TQBF, construct the circuit as in Theorem 6, but make an extra copy of the circuitry for the threshold. In this copy, insert a MAX gate in the region containing the last quantifier gadget, and pass the gadget's output into the second input of the MAX gate, constructing the register circuit shown in Figure 9. Suppose we otherwise treat this copy exactly as we treat the threshold—in particular, suppose we update its AVG gate for the transition into the first region of the circuit immediately after the AVG gate for the threshold in the first region (i.e. make it gate index 2). Denote the circuit constructed in this way by C and consider the problem LEAPFROG-LIMIT$(C, 2, 1)$.

Suppose $\phi \notin$ TQBF. Then, as we argued before, on every round the output of the final quantifier gadget is below threshold. Thus, on every round the input to the register will be below threshold, so the register circuit's internal wire will stay at threshold, and hence the register will continue to mimic the threshold circuitry precisely. Therefore, LEAPFROG-LIMIT$(C, 2, 1)$ is a "no."

Now, suppose instead $\phi \in$ TQBF. Then, on some tth round, the output of the final quantifier gadget is above threshold. Hence, the input to the register circuit will also be above threshold, so a value greater than threshold will be placed on the internal wire. Then, since the action of the AVG gates during translation between regions is strictly monotone, the internal register wire will stay strictly above threshold. Thus, clearly for all $n > t$, $v_2 > v_1$, and hence LEAPFROG-LIMIT$(C, 2, 1)$ is a "yes" instance, and since this extended construction should also be easily performed in logspace, we have logspace-reduced TQBF to LEAPFROG-LIMIT. □