6.890 Lecture 22 - Scribe Notes

Today's lecture is a guest lecture by Costis Daskalakis, a world leading expert in algorithmic game theory.

In 1928, von Neumann published his minimax theorem [1], which states that any zero-sum game will have a Nash equilibrium (although at the time this was not called a Nash equilibrium). In the 1950s, Nash extended this result to general games [2].

Interestingly enough, this small difference is a huge gap on the algorithmic side. The first problem can be solved in polynomial time via LP duality, whereas the more general problem has no known polynomial time algorithm. This dichotomy provides the motivation for the PPAD class.

In this lecture, we will introduce three key theorems that are interrelated and are PPAD-Complete. We will also formally define the PPAD class and sketch the proof that the general Nash equilibrium problem is PPAD-Complete. The full proof will be shown on the next lecture.

1 Three Key Existance Theorems

We will analyze existence theorems by Nash, Brouwer and Sperner that arise from game theory, topology and combinatorics, respectively. These three theorems, as disimilar as they seem, can be related to one another and rely on very basic combinatorial principles.In this section, we will define the setting for each of the theorems and give an overview of their connection.

The motivation to study these problems is that, unlike many problems in computer science, we are told that a solution exists. How efficiently can we compute this solution? If the search space is polynomial, this is easy: simply search exhaustively. However, some of these problems can be defined succintly and have exponential size. In these cases, the search problem requires a larger framework.

1.1 Nash Equlibria

Definition A finite game consists of the following elements:

- A set of n players $p \in P$.
- A set of strategies S_p for every $p \in P$.
- A utility or payoff function that assings a real value to player p for every possible strategy set. Formally, we need $u_p : \times S_i \to R$, for every $p \in P$.

For the purposes of this lecture, we will be interested in what is known as Nash equilibrium (NE). A collection of pure strategies $s_1, s_2, ..., s_n$ is a NE if for every player $p \in P$,

$$u_p(s_1, s_2, ..., s_p, ..., s_n) \geq u_p(s_1, s_2, ..., s'_p, ..., s_n) \forall s'_p \in S_p$$

Informally, this definition says that a strategy is NE if no player has incentive (i.e., can't be better off) to change his strategy based on the strategies of the other players.

Unfortunately, many games don't have pure NE. However, if we allow players to pick their strategy according to some distribution we can prove there are always NE. This motivates the use of randomization. In particular, a randomized strategy x_p is one where a player picks a distribution Δ over their various strategies $\Delta(S_P)$. We can extend our notion of NE to randomized strategies by taking expected values based on the distributions.

In 1928, von Neumann showed that in two-player zero-sum games there is always an equilibrium [1]. This minimax theorem turned out to be a special case of strong LP duality. This implies that NE can be computed in polynomial time. In 1950, Nash showed equilibrium exists in general games [2]. Unfortunately, there is no known LP proof or polynomial time algorithm to compute such equilibria.

As an example, consider the following game: Player 1 needs to decide where to shoot a penalty (left or right) and Player 2 needs to decide where to dive (left or right). The payoff of this game is easy to describe: if Player 1 scores, he gets a point and player 2 gets -1 points. If Player 1 misses, Player 2 gets a point and Player 1 gets -1 points.

There is no pure NE in this game. Consider any strategy. If the two strategies matched, player 1 would have incentive to deviate. Otherwise, player 2 would have incentive to deviate. However, there is a mixed NE: each player flips a fair coin. If it is heads, he goes right and if it is tails, he goes left. This can be shown to be a NE.

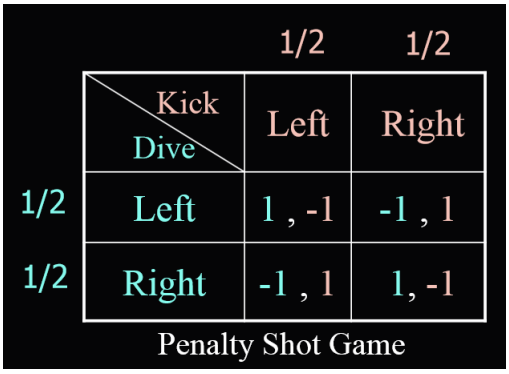

Figure 1: Kick Dive payoff matrix.

1.2 Brouwer's Fixed Point Theorem

Theorem 1.1. *(Brouwer's Fixed Point Theorem [3]) Let $f : D \to D$ be a continuous function from a convex and compact subset D of Euclidean space to itself. Then, there exists $x \in D$ such that $f(x) = x$.*

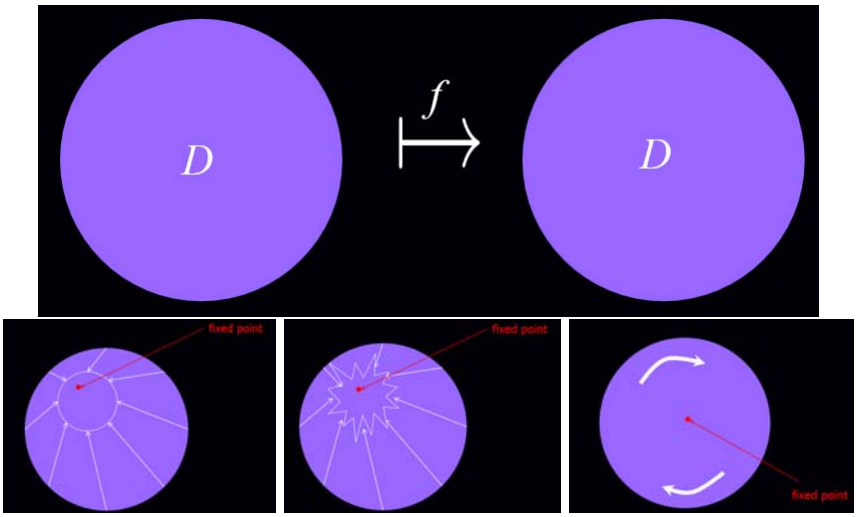

Figure 2: Brouwer's fixed point theorem is a mapping from D to D.

In particular, this theorem is tight: if any of the conditions are not met, the theorem is not true.

It is worth noting that Nash used Brouwer's theorem to show his result for general games. Roughly, the proof involves a function $f : [0,1]^n \to [0,1]^n$

as a vector field that indicates the motivation a player has to deviate from his current strategy. The NE corresponds to the fixed point of the mapping. The following picture depicts the function for the game we studied in the previous section:

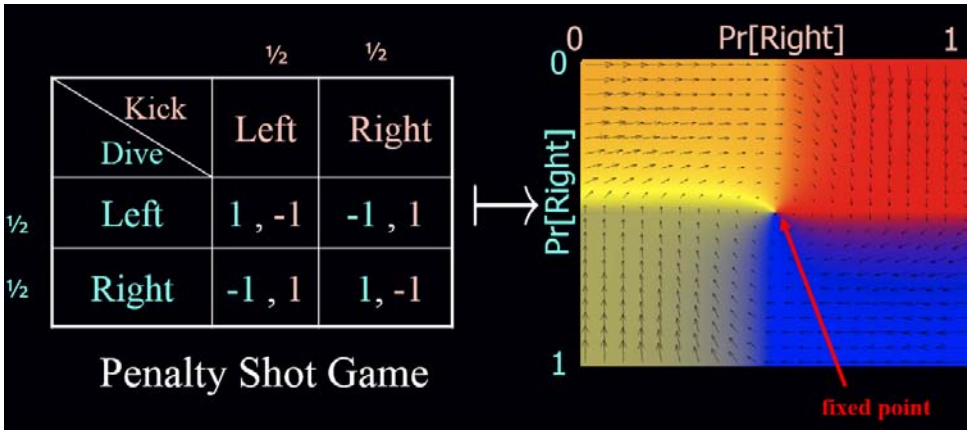

Figure 3: The proof of Nash using Brouwer.

1.3 Sperner's Lemma

We will be looking at the special case where the dimension is 2, but there is a natural generalization of the Lemma.

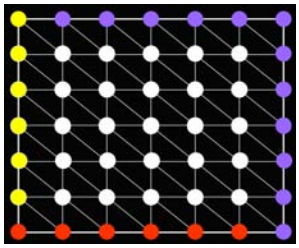

Figure 4: Valid edge coloring for the Sperner edge coloring.

Theorem 1.2. *(Sperner [4]) Suppose we are given a triangulized square grid graph and three colors: red, blue and yellow. Let a legal boundary be one in which all nodes in the bottom line of the grid are red, all nodes in the leftmost column are yellow and all other boundary nodes are blue. Then,*

for every coloring of the internal nodes, there will always be a trichromatic triangle. In fact, there will be an odd number of them.

There is also a natural connection between Sperner's and Brouwer's theorems. We can consider the problem of finding approximate fixed points $|f(x) - x| < \epsilon$ for some $\epsilon > 0$. We will create a mesh on the space of the function and color each node according to the direction of $f(x) - x$ in one of three colors. We can then use a compactness argument and let $\epsilon \to 0$ to prove the existence of a fixed point (which will be inside the trichromatic triangle). This proof however might not preserve the parity or even the number of trichromatic triangles.

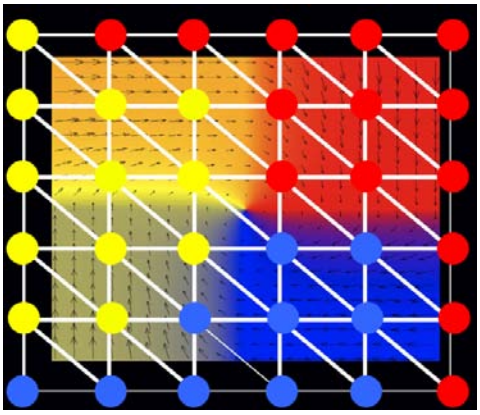

Figure 5: The overlay of Sperner's coloring on a function mapping $[0,1]^2 \to [0,1]^2$.

1.4 Proof of Sperner

We will now provide a proof of Sperner's Theorem and highlight the mathematical principle that underlies it.

First, we will add an artificial trichromatic triangle by adding a blue vertex next to the bottom left corner of the grid, where the yellow and red boundaries meet. We now define a directed walk on the triangles of the grid graph. While we haven't found a new trichromatic triangle, cross a yellow-red edge, with the yellow node to the left of the path. We claim that this procedure will find another trichromatic triangle.

Note that this walk can not exit the grid graph with legal boundary coloring: the only red-yellow edge on the boundary is the one on the bottom left corner, and in order to cross it we would have to have the yellow node

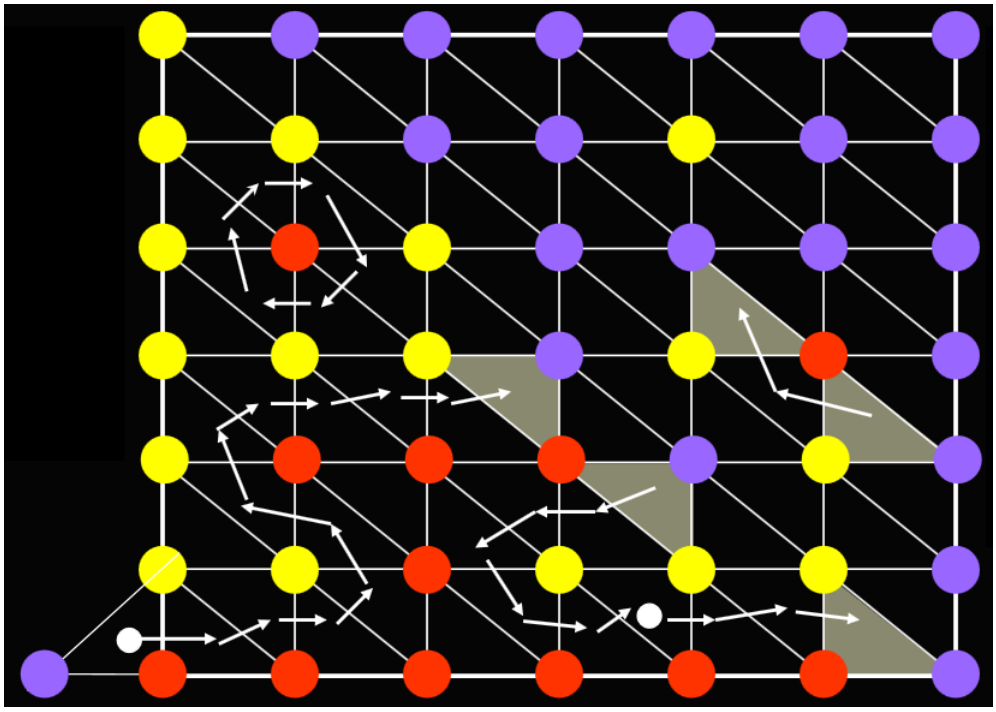

Figure 6: A valid walk in a valid coloring of a triangulation of the lattice for Sperner's lemma.

on our right. This is not allowed. Moroever, our walk will not produce a cycle. For the sake of contradiction, suppose that it did. Consider a triangle where the loop closes. This triangle must have had a red-yellow edge that we crossed the first time with yellow to the left. However, on our way back in, any edge we cross will either have red to the left or yellow to the right. Neither of these options is admissible. Therefore, there are no cycles and we never leave the grid graph.

This, together with the fact that the number of triangles is finite, implies that at some point we must encounter a trichromatic triangle, since that is when our walk stops. Therefore, at least one such triangle must exist.

What about any other trichromatic triangles? Well, we can perform the same procedure with the other internal trichromatic triangles. Start another walk from one such triangle and, by the same argument above, we will end at another.

Therefore, the total number of trichromatic triangles in our modified graph is an even number greater than 2. However, one of these triangles

was artificially introduced. Therefore, the number of trichromatic triangles inside the grid graph must be odd and greater than 1.

1.5 A more basic approach

There is yet another more basic proof of this. Consider a directed graph where each node represents a triangle. There is an edge (u, v) if triangles u, v are adjacent and the edge that they share is a 'crossable' edge (i.e. yellow on the left).

We claim that every vertex must have either in-degree or out-degree at most 1. This can be done by an exhaustive analysis of the possibilities. It is important to note that if a triangle has exactly one red-yellow edge (hence it's tri-chromatic), then it's node will have either exactly in-degree 1 and out-degree 0 or out-degree 1 and in-degree 0.

But what can we say about a directed graph where very node has in-degree and outdegree at most 1? Well, there can only be three types of (weakly) connected components: isolated nodes, cycles or directed paths. These paths correspond to the paths we discovered in the walk above and imply that the trichromatic triangles come in pair. In a more elementary way, the underlying principle is that if a directed graph has an unbalanced (i.e. indegree is not the same as out-degree) node, then there must be another one.

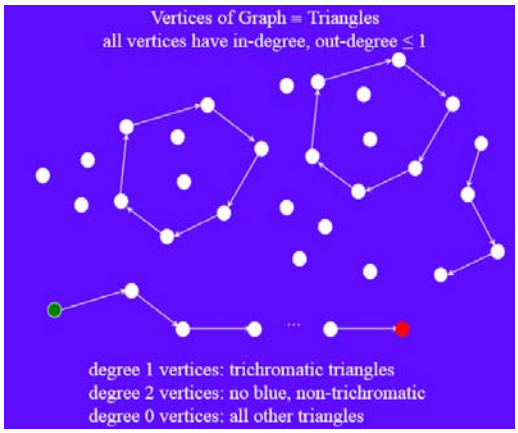

Figure 7: The triangulation of Sperner's lemma turned into a directed graph.

In our example, the unbalanced nodes correspond exactly to trichromatic triangles. Since in our construction we introduce one such node, we are guaranteed that our graph will contain another one and a total even number

of them. This property is also known as the Parity Argument for Directed Graphs, and plays a key role in the definition of the PPAD class.

2 Total Search Problems in NP

For a search problem, we want to find an instance of a given property. For a total search problem we are guaranteed that at least one such instance exists, but, we must return one such instance. For Nash, Bouwer and Sperner what we care about are search problems, usually over polynomially describable infinite search spaces.

2.1 FNP and TFNP

We will first define search problem.

Definition A search problem L is defined by a relation $R_L \subset \{0,1\}^* \times \{0,1\}^*$ such that $(x, y) \in R_L$ if y is a solution to x.

A search problem is total if every problem has a solution.

Definition A search problem L is total if $\exists y \; \forall x$ such that $(x, y) \in R_L$.

2.1.1 FNP definition

To get an intuition for FNP let us first define FP.

Definition [5] A problem L is in FP if there exists a polynomial time algorithm that given an x can find a y such that $(x, y) \in R_L$.

Instead of deciding if there exists such a y, as we do in P, we must return such a y for problems in FP.

FNP will relate to FP in a similar way to how NP relates to P. FNP is the class of problems where search problems can be verified efficiently. If we can verify x, y in R_L for a y that is at most polynomially sized relative to x in polynomial time, then the problem is in FNP.

Definition A problem L is in FNP if

1. There exists a polynomial time algorithm that given an x and a y can check if $(x, y) \in R_L$.

2. There exists a polynomial $p_L()$ such that: given an x the existence of a y such that $(x, y) \in R_L$ implies that $\exists z$ where $|z| \leq p_L(|x|)$ and $(x, z) \in R_L$.

Note that the first condition corresponds to being able to verify answers efficiently. The second condition guarantees that for any x where there exists a y such that $(x, y) \in R_L$ there will exist some polynomial length z that satisfies $(x, z) \in R_L$.

TFNP definition

We want to capture the fact that the problems we care about are total in a new definition.

Definition TFNP $= \{L | L \in$ FNP and L is total$\}$

Note that the existence of a solution and the polynomial condition of FNP result in the following equivalent definition.

Definition A problem L is in TFNP if

1. There exists a polynomial time algorithm that given an x and a y can check if $(x, y) \in R_L$.

2. There exists a polynomial $p_L()$ such that $\forall x\ \exists y$ such that $(x, y) \in R_L$ and $|y| \leq p_L(|y|)$.

FNP reductions

At first it seems we would like to show the search problem of SPERNER to be FNP hard. Sadly, it doesn't seem likely that we can reduce these search problems to FNP.

Definition A problem $L \in$ FNP is poly-time Karp reducible to $L' \in$ FNP if:

1. There exists an efficiently computable f such that: $\{0,1\}^* \to \{0,1\}^*$ maps inputs x to L into inputs $f(x)$ to L'

2. And an efficiently computable g such that:

$$\forall x, y : A_{L'}(f(x), y) = 1 \Rightarrow A_L(x, g(y)) = 1$$

$$\forall x : A_{L'}(f(x), y) = 0, \forall y \Rightarrow A_L(x, y) = 0, \forall y$$

Where A_L is an algorithm that solves L and $A_{L'}$ is an algorithm that solves L'.

Current evidence suggests we won't be able to show NASH, SPERNER and BROUWER to be FNP hard. We can't reduce to NASH, SPERNER, nor BROUWER from SAT. If we can reduce to these problems from the Turning Machine Problem, then it implies coNP =NP. Basically, these are a different kind of problem. We know these problems are satisfied, which makes reducing to a problem where the core complexity comes from determining satisfiability infeasible.

This is what motivates the PPAD (Polynomial Parity Arguments for Directed graphs) class, originally introduced by Papadimitriou in 1994 [6].

3 PPAD definition

We will define a problem that is motived by an existence proof. Specifically, the proof of Sperner's Lemma uses the fact that nodes with max in and out degree of 1 result in paths. It relies on the fact that a directed graph with an unbalanced node implies another unbalanced node. As pointed out before, this type of argument is refered to as Parity Argument for Directed Graphs.

We will define a problem based on this existence proof.

3.1 END OF THE LINE

Suppose that an exponentially large graph with vertex set $\{0,1\}^n$ is defined by two circuits:

Form a graph where $E = \{(v_1, v_2) | P(v_2) = v_1 \wedge N(v_1) = v_2\}$.

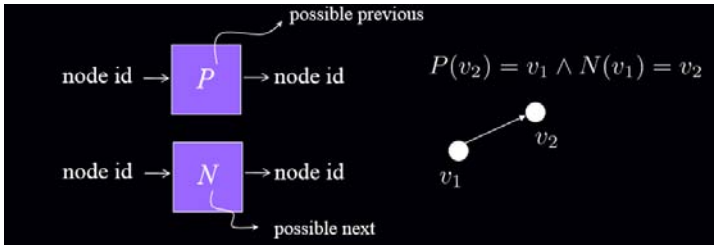

Figure 8: The input to the END OF THE LINE problem.

Given P and N: if 0^n is an unbalanced node, find another unbalanced node. Otherwise output 0^n.

We know that the circuit described by P and N has max out degree 1 and max in degree 1. Notice, this looks exactly like the constraints for Sperner's Lemma!

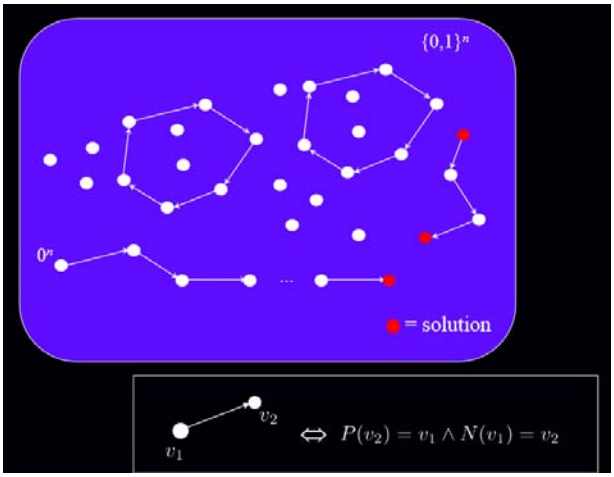

Figure 9: The graph produced by the END OF THE LINE input.

Further note that checking if 0^n is unbalanced takes $O(1)$ time. We check if $N(P(0^n)) = 0^n$ and if $P(N(0^n)) = 0^n$ if one is and one isn't then we have that 0^n is unbalanced. Given that 0^n is unbalanced there must exist an unbalanced node somewhere else in the graph.

3.2 PPAD-Completeness

Something is PPAD-hard if it can solve END OF THE LINE. A problem is PPAD-complete if it is also solvable by END OF THE LINE.

4 PPAD-complete problems

END OF THE LINE, NASH, SPERNER and BROUWER are all PPAD complete.

4.1 PPAD-completeness of NASH (high level)

We first find cycles and paths and place them in a cube $[0, 1]^3$ without intersections. Then we represent this as a SPERNER problem, next we convert the problem to an Arithmetic Circuit. Finally, that Arithmetic Circuit is converted into a NASH problem.

The full proof was part of Daskalakis' PhD Thesis and is based on joint work with Goldberg and Papadimitriou [7].

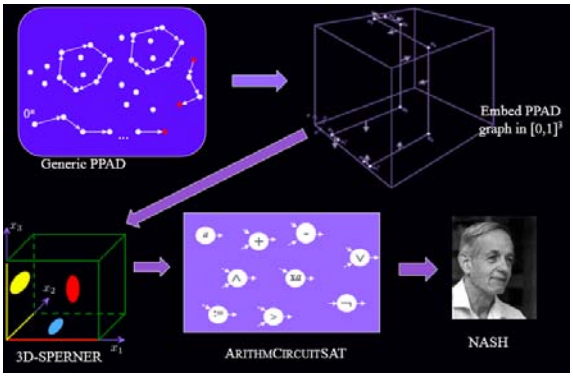

Figure 10: A high level overview of the proof that NASH is PPAD hard.

4.2 ArthmCircuitSAT

Input is a circuit comprising

- variable nodes $x_1, x_2, \ldots, x_n$
- gate nodes $g_1, \ldots, g_m$ of 6 types $(:=, +, -, a, xa, >)$
- directed edges connecting variables to gates and vice versa
- Variable nodes have in-degree 1; gates have in degree 0,1, or 2 inputs depending on type ; gates and nodes have arbitrary fan out

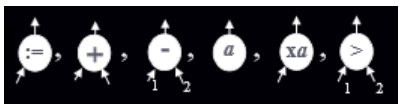

Figure 11: The operation nodes of ArthmCircuitSAT.

The output is a set of values satisfying the circuit constraints.
The types of gate nodes are:

- Assignment: $y == x_1$
- Addition: $y == \min\{1, x_1 + x_2\}$
- Subtraction: $y == \max\{0, x_1 - x_2\}$
- Equal a constant: $y == \max\{0, min\{1, a\}\}$

- Multiply by a constant: $y == \max\{0, min\{1, ax\}\}$
- Greater than: $y == \begin{cases} 0, & \text{if } x_1 < x_2 \\ 1, & \text{if } x_2 < x_1 \\ \text{any value}, & \text{if } x_2 = x_1 \end{cases}$

Arithmetic circuit satisfiability will be used to show that NASH is PPAD hard.

4.2.1 Example

In Figure 4.2.1 we show an example of ArthmCircuitSAT.

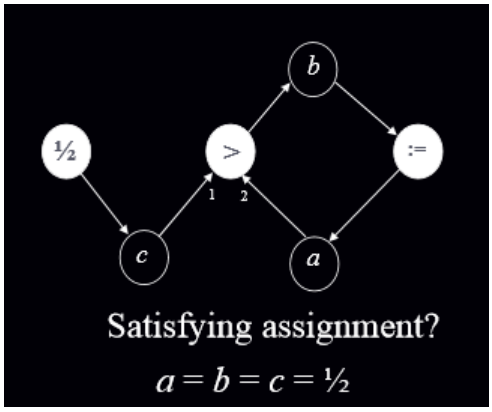

Figure 12: The operation nodes of ArthmCircuitSAT.

Node c is assigned to value $1/2$ and thus must have this value.

If $a > c = 1/2$ then b will be set to 0, $a = b$ and $0 < 1/2$ so this is a contradiction.

If $a < c = 1/2$ then b will be set to 1, $a = b$ and $1 > 1/2$ so this is a contradiction.

If $a = c = 1/2$ then b can be set to any value, $a = b = c = 1/2$.

References

[1] Von Neumann, J: Zur Theorie der Gesellschaftsspiele Math. Annalen. 100 (1928) 295-320

[2] Nash, John: Equilibrium points in n-person games. Proceedings of the National Academy of Sciences 36 (1950):48-49.

[3] L.E.J. Brouwer: Ueber eineindeutige, stetige Transformationen von Flchen in sich. Math. Annalen. , 69 (1910) pp. 176180

[4] Sperner, E. : Fifty years of further development of a combinatorial lemma. Numerical solution of highly nonlinear problems. Sympos. Fixed Point Algorithms and Complementarity Problems, Univ. Southampton, Southampton, 1979: Part A, p.183197, Part B, p.199214.

[5] Rich, Elaine: Automata, computability and complexity: theory and applications. Prentice Hall, 2008, ISBN 0-13-228806-0, section 28.10 "The problem classes FP and FNP", pp. 689-694

[6] Papadimitriou, Christos: On the complexity of the parity argument and other inefficient proofs of existence. Journal of Computer and System Sciences 48 (1994): 498532

[7] Daskalakis, C., Goldberg, P., Papadimitriou, C.:The Complexity of Computing a Nash Equilibrium.Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing. STOC 2006, ACM, pp. 71-78

6.890 Algorithmic Lower Bounds: Fun with Hardness Proofs

Fall 2014