

Lecture 18 Scribe Notes

Prof. Erik Demaine

1 Overview

This lecture looks at more games, and so far, we have been focused on 1 player games, or puzzles. Instead, we will be focusing on 0-player games, or simulations, and 2 player games. In each case, if the game is on a bounded board, then it will either take an exponentially bounded number of moves (unbounded) or a polynomially bounded number of moves (bounded). Generally speaking, these games are in the complexity classes given by Figure 1. We will look at hardness results for several games in particular, including

- Conway’s Game of Life, a 0-player game with an unbounded number of moves
- Deterministic constraint logic (DCL), a 0-player game with an unbounded number of moves
- Various SAT games, all 2-player games with a bounded number of moves
- Reversi/Othello, a 2-player game with a bounded number of moves,

All of these games are in fact PSPACE-complete.









unbounded				
	PSPACE	PSPACE	EXPTIME	Undecidable
bounded				
	P	NP	PSPACE	NEXPTIME
	0 players (simulation)	1 player (puzzle)	2 players (game)	team, imperfect info

Figure 1: A table showing the complexity classes for each type of gam, with bounded and unbounded referring to whether the number of moves is polynomially bounded or not.

Now, we might also consider games played on infinite boards, and a result shows that Conway’s Game of Life, which is a simulation, is undecidable when played on an infinite board. Lastly, if a simulation is on a finite board, with a polynomially bounded number of moves, then it must be in P, given that each move is deterministic and takes polynomial time.

2 Conway's Game of Life

In Conway's game of life, we have a grid of cells that are either living or dead. A cell's neighbors are defined to be the eight cells that are horizontally, vertically, or diagonally adjacent to it. Each iteration of the simulation, the cells update as follows:

1. A living cell stays alive if it has 2 or 3 living neighbors. Otherwise it dies.
2. A dead cell becomes living if it has exactly 3 living neighbors.

With these rules, we can end up with periodic patterns, e.g. pulsars, or static patterns, known as still life. Some of these can be seen in Figure 2. For more examples of how various patterns behave, there are many interactive simulations online.

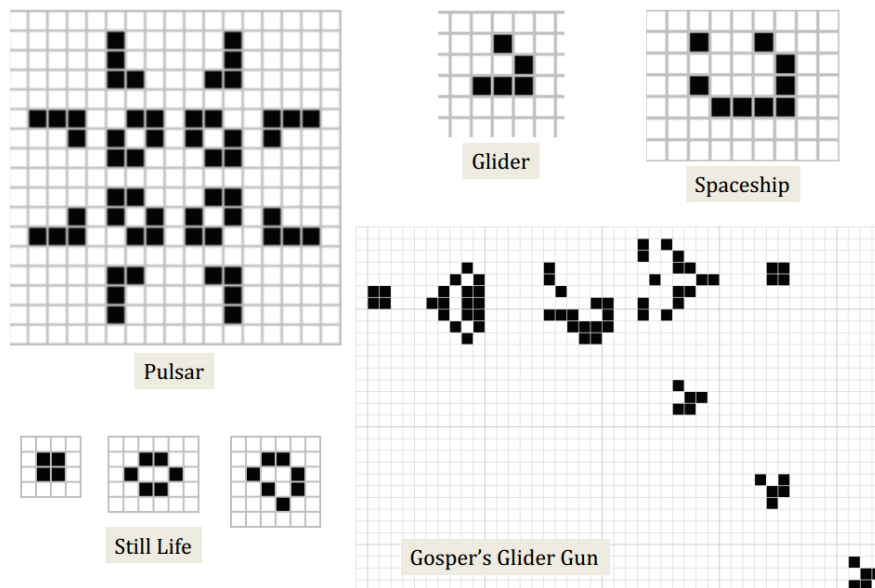


Figure 2: A couple of common patterns in Life. Black squares are living, and white squares are dead.

There are a couple different decision problems we can ask. The first, whether it is an example of still life is easy; we just compute the next step. We can also ask whether the configuration is periodic and whether all the cells will die out, which are a bit harder. These are unbounded simulations, and with polynomial space, they are PSPACE-complete, whereas with infinite space but a finite starting area for live starting area, they are undecidable.

2.1 Turing machine simulations

These results are generally proven by showing that Conway's Game of Life can simulate Turing machines. For example, Figure 3 shows a turing machine simulation that takes a number in unary and doubles it (Paul Randell 2000). For finite space boards, this proves PSPACE-hardness, as any decision problem in PSPACE can then be solved by creating the proper Turing machine simulation.

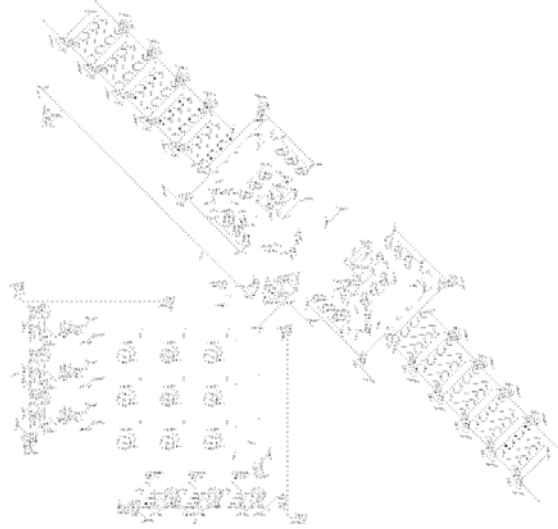


Figure 3: An example of a turing machine simulation in Life.

When given infinite space, we can create universal turing machines, which are turing machines that can simulate arbitrary turing machines acting on arbitrary input. This requires however, an infinite tape, which can be obtained by creating a turing machine that grows itself, constructing a longer and longer tape as needed. Determining whether this universal turing machine grows infinitely or not is undecidable, since it is equivalent to the halting problem. However, this method does not prove anything about the question of whether all the cells eventually die out. So, because most of the decision problems are similar, we focus on the proof of undecidability of whether we eventually reach the all-dead state.

2.2 Life is Undecidable

The gadgets used in this construction rely on a few key patterns in the Game of Life, seen in Figure 2 and Figure 4:

- Glider - this is a periodic pattern that travels diagonally off to infinity.
- Glider gun - this pattern produces gliders at some constant rate.
- Eater - this pattern annihilates gliders that hit it at the proper offset.

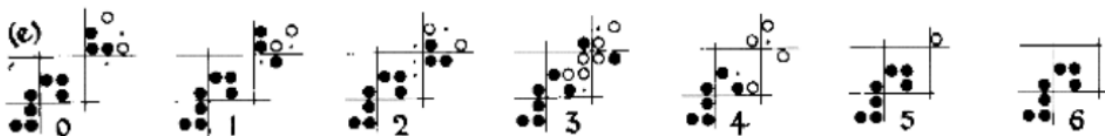


Figure 4: A diagram showing an eater annihilating a glider. Circles represent living squares, white circles represent living squares about to die, and dots (which may be hard to see) represent dead squares about to come to life.

We can represent streams of bits using streams of gliders, where a glider represents 1, and a missing glider represents a 0. When two gliders collide with the right offset, they annihilate each other. In this way, we can create an inverting turn gadget by having a glider gun continuously shoot at our stream, as represented by Figure 5. This allows us to delay a signal by performing multiple turns in a row, which is important since the timing of signals is important. This however, does not change the offset of a signal, but since gliders at a number of different offsets can annihilate each other, a glider gun can be used to perform a parity shift.

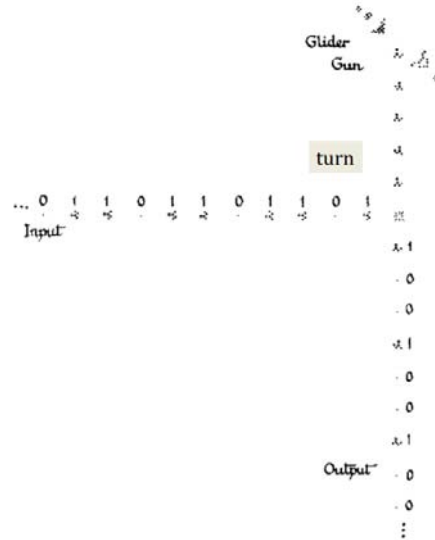


Figure 5: An inverting turn gadget. A glider gun at the top shoots gliders towards the output, which passes only if the input bit was a 0. Otherwise the input glider annihilates the gun's glider.

Using a combination of glider guns and eaters, we can then create AND and OR gadgets, as in Figure 6.

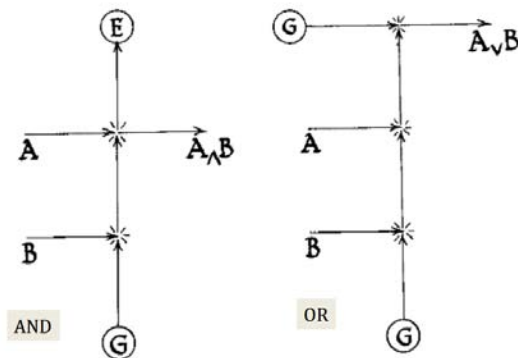


Figure 6: An AND gadget and an OR gadget. All timing issues can be solved by adding turns to delay input. AND: if B is 0, then gliders from the gun pass through and annihilate A. OR: For a glider at the top to not be annihilated, only one of A or B needs to block gliders from the bottom.

We can also create a kickback, in which two gliders collide at the right offset, reversing one and annihilating the other (Figure 7). Using two kickbacks to send a glider back and forth between

two streams, we can add holes to the streams at a rate determined by the distance between them. Inverting one of the streams with a glider gun lets us thin out the stream by lowering the glider rate. By thinning out two crossing streams, we can obtain crossovers by offsetting them so that the gliders don't collide.

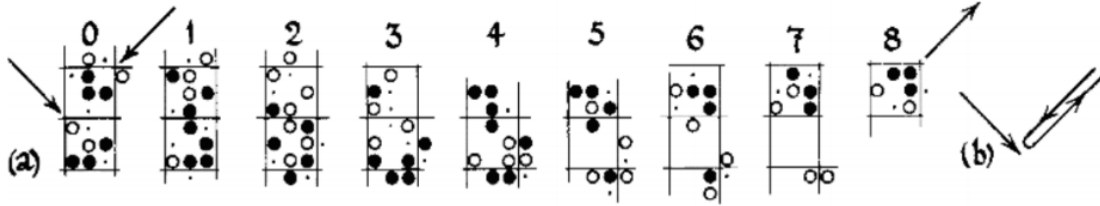


Figure 7: (a) A diagram showing a glider being kicked back, with the other glider being annihilated. (b) A kickback is often represented with this diagram.

Finally, with kickbacks and OR Gadgets, we can create a SPLIT/NOT Gadget. The full details are given in the caption for Figure 8, but the general idea is to send a glider along with each data bit in a low-density stream, and they interact with a full stream to produce two copies and an inverted copy. This gives us a direction preserving NOT.

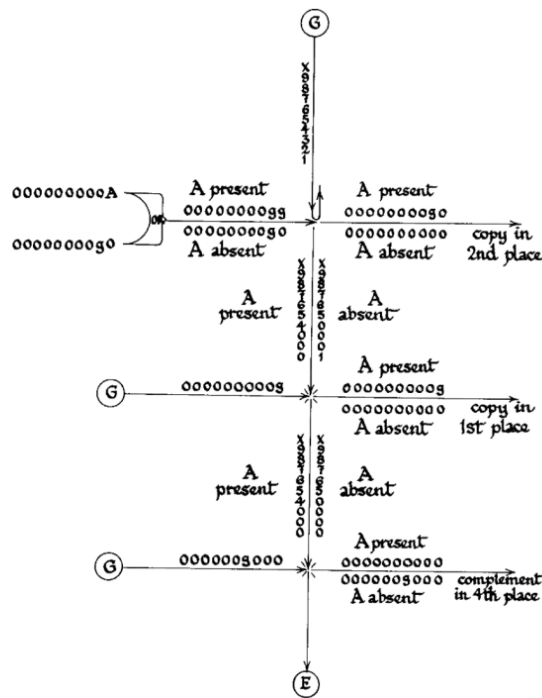


Figure 8: Kicked back gliders annihilate two other gliders (three total). Each data bit A is sent with an additional glider g , one is annihilated kicking back a glider in the full stream coming from the top. So, the resulting stream is our first copy, with the data bit in the second place. Then, the bottom two streams test for gliders in the first and fourth bit, giving another copy and an inverted copy.

With these gadgets, we now have a finite computer. However, to prove undecidability, we would need registers that can store arbitrarily large integers, which can be incremented, decremented, or checked for equality with 0. So, we simulate a Minsky machine, which is a counter machine with only 2 registers, and a Minsky machine is sufficient to simulate an infinite turing machine [6].

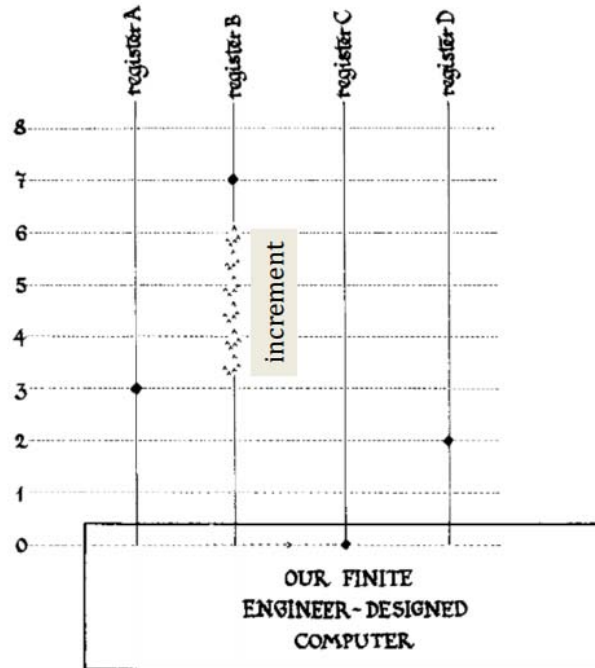


Figure 9: A 2-counter machine in Life. A wave of gliders increments an integer register, and a single glider (near the bottom) tests equality with 0.

A representation of the 2-counter machine is given in Figure 9. We use a 2 by 2 box to represent the register. The position of the box represents the register value, which can be unbounded. We can test if a box is in position 0 by shooting out a glider. If the box is there, the glider is destroyed, along with the box. Otherwise, the glider passes through. If the box is destroyed by the check, then we can recreate the box using 2 gliders with the right offsets. No single glider can increment or decrement a register, but there is a way to send a wave of gliders to push the block up by 1, or pull it back by 3. We can then combine these to give arbitrary increment or decrements.

Of course, a wave of gliders requires guns that are close together, but glider guns are large and cannot be close together. Instead, we can use three computer-controlled glider guns to shoot gliders along parallel paths offset from each other, as in Figure 10. Sending these gliders along parallel paths is enough to perform increment and decrement.

Lastly, we need our Minsky machine to self-destruct after halting. To do this, we use a boomerang gadget, which can send a glider arbitrarily far out and then bring it back using kickbacks. We can now configure it so that if the machine says yes, we use a boomerang gadget to send out a bunch of gliders and then bring them back at the right angles to destroy all the gadgets that we have, leaving us with a completely empty board.

This proves the undecidability of life.

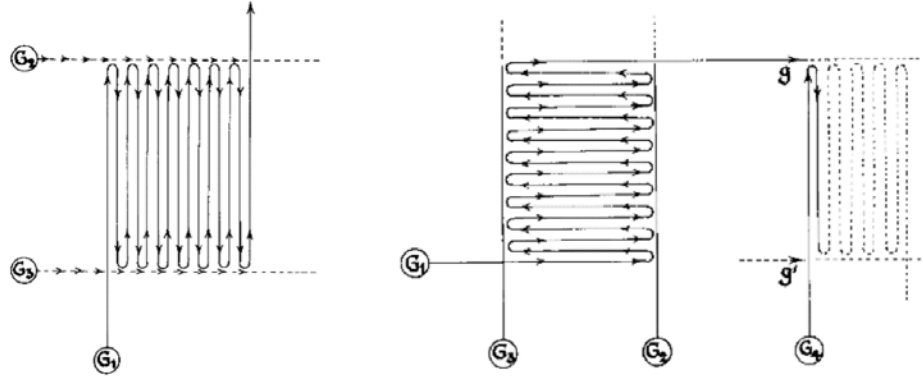


Figure 10: Left: A positioning mechanism with which gliders can be shot with an arbitrary offset. Right: a boomerang gadget, that can send a glider arbitrarily far up, then bring it back down, without placing any guns above a certain line.

3 Deterministic Constraint Logic

We now add rules to constraint logic to ensure that all moves are deterministic, rather than chosen by a player. Aside from orientation of edges, we also track whether an edge is active or inactive. An edge is active if it was just flipped in the last round. Otherwise, it is inactive. We also say a vertex is active if its active incoming edges have a total weight at least 2.

In each round of DCL, the following things happen:

- Inactive edges pointing to active vertices are reversed
- Active edges pointing to inactive vertices are reversed.
- The edges that have been reversed are the new active edges.

The question associated with an instance of DCL is whether an edge in a given graph can ever be reversed. Then, DCL is in PSPACE, since we only need to keep track of the current state. So, in order to show PSPACE-completeness, we reduce from Q-CNF-SAT, as shown in Figure 11.

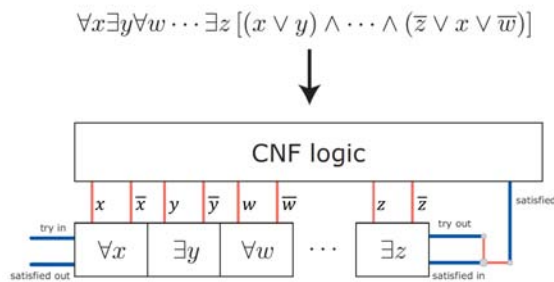


Figure 11: DCL Gadget for QBF instance. If we can find a solution for the QBF formula, then we can reverse the edge labelled “satisfied.”

Our key gadget for DCL is the switch gadget, as shown in Figure 12. For this gadget, we place a guarantee that the inputs must reverse at times $t = 0 \pmod{4}$ to deal with parity issues. The switch acts like a splitter. When the input A flips, the activation propagates down to B . Later, when B flips back, the activation propagates to C instead. When C flips back, B is activated a second time. When B reverses back, then A reverses and we exit the gadget. If we consider B as a variable x , we see that the switch essentially tries x , then not x , and x again for good measure.

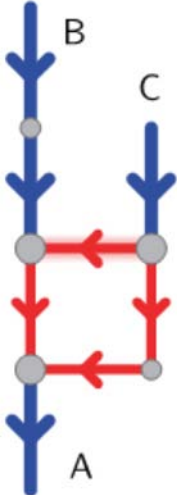


Figure 12: A DCL switch gadget. Activated edges are highlighted.

Using our switch gadget, we can build an existential quantifier which activates an output edge if either x or $\neg x$ can satisfy. Similarly, we can build a universal quantifier which activates on the output if both x and $\neg x$ can satisfy. These gadgets are seen in Figure 13. Essentially, they test each value of a given variable, and acknowledge the resulting value on another edge. We use degree-2 vertices to fix timing.

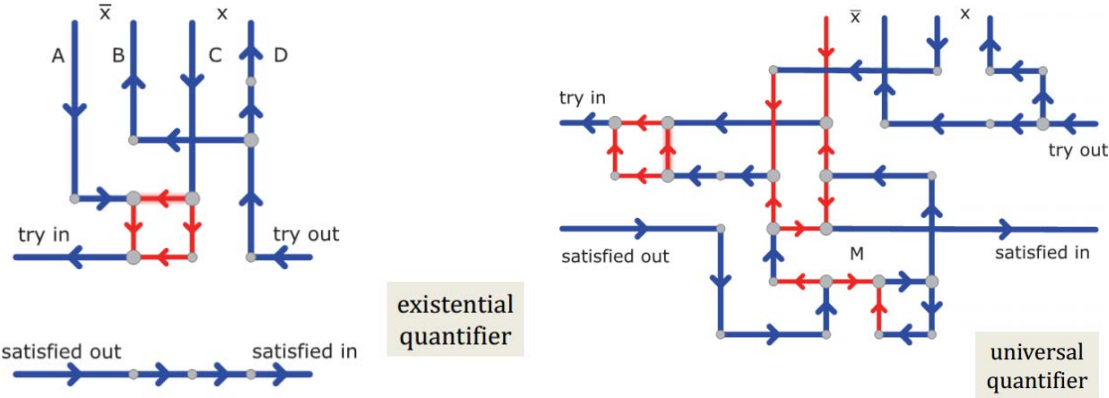


Figure 13: An existential and universal quantifier. Acknowledgement that the circuit is satisfied is passed down the wires marked satisfied in and satisfied out.

Using the original gadgets for nondeterministic constraint logic results in timing issues, as they only work if all variable signals arrived simultaneously. For DCL, we use modified gadgets, AND'

and OR', with 2 channels for each input, as shown in Figure 14. The first channel is an activation signal, while the second channel serves as an acknowledgement. For the AND' gadget, we use the switch to hold a signal until both inputs are received and only send back an acknowledgement after this. Using similar constructions, we can also get OR' and split gadgets.

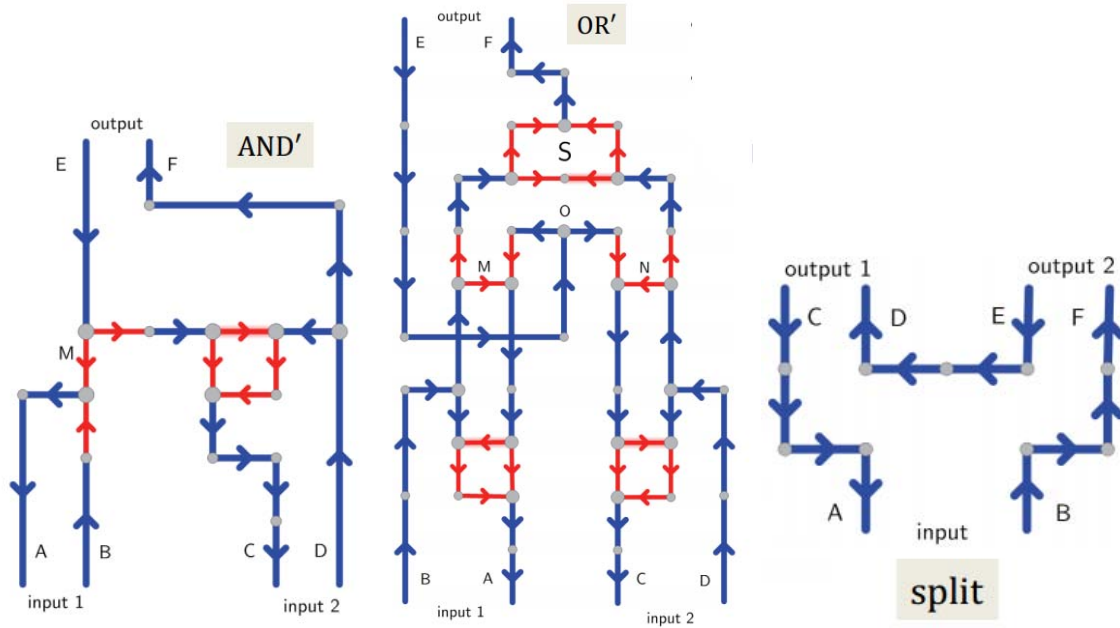


Figure 14: The AND', OR', and SPLIT gadgets.

To finish the construction, we split all wires into 4 to ensure correct timing and use new gadgets which maintain the 0 (mod 4) timing to deal with the resulting degree-2 vertices. We can then construct a crossover gadget (Figure 15) to show hardness for planar graphs, too.

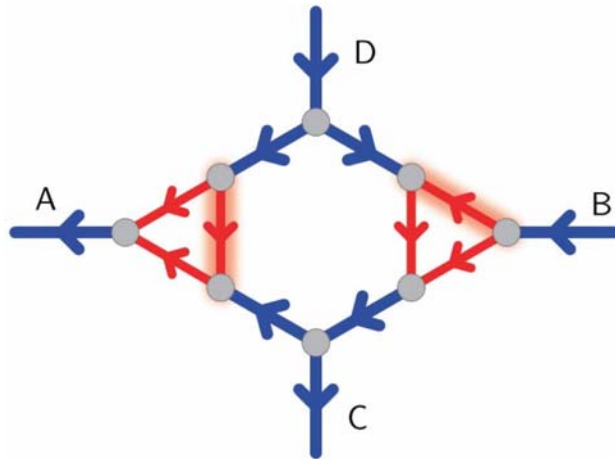


Figure 15: The crossover gadget for DCL. Signals pass right through, so long as the timing, which we can control, is correct and no two inputs are activated at the same time.

4 Two Player Games

After considering zero player games, or simulations, the natural question is whether multi-player games can also be considered within the complexity bounds of PSPACE. Typically, these questions are posed as “Does the first player have a winning strategy that forces a win?” or “Given a game position, can the next-player-to-move force a win?”

This question can be reduced to the question of whether a player has a winning strategy for a two-player version of the game. In the multiplayer version, we could consider the $n - 1$ other players as a “hive-mind” that colludes to prevent player 1 from winning. This situation would be asymmetric with respect to player 1 because the adversary would have 2 or more moves for each move of player 1, but it demonstrates that in the worst case, the game can be considered a two-player simulation where one player simply has much greater power.

Therefore we generally consider the complexity of two player games. These players can be labelled “black” and “white” (as they are for games such as Chess, Go, etc), and we consider the complexity of the game given a polynomial number of moves, necessarily restricting the game to PSPACE, since all the moves and the results can be described using a polynomial amount of space. A player wins if they can find a set of winning moves that handles all responses by an opponent that satisfies the formula $\exists \text{ move: } \forall \text{ responses: } \exists \text{ move: } \forall \text{ responses...}$ Player 1 wins. Winning the game is similar to finding a solution for a specific instance of QSAT.

4.1 SAT Games

SAT-style games were defined by Schaefer in [9]. These QSAT games take use a set of quantifiers \exists and \forall and a SAT formula, where player 1 tries to satisfy the formula, and can assign all odd variables. Player 2 tries to “break” the formula by assigning the values of the even variables.

There are variations on this game in which different players moves may be restricted. The first variation is an impartial game and the second is a partizan game, defined as:

- Impartial Game: On a turn, a player may set the value of any unassigned variable. (Both players may make the same moves)
- Partizan Game: Variables are categorized as white variables or black variables, with equal numbers of both. On each turn a player sets the value of any unassigned variable of the same color.

Schaeffer also describes three versions of a QSAT game with different definitions of a win.

- (Default) Game: Player 1 wins \iff formula is satisfied at the end.
- Seek Game: The first player to satisfy the formula wins. (All unassigned variables have value \emptyset .)
- Avoid Game: The first player to satisfy the formula loses. (All unassigned variables have value \emptyset .)

Then the following games are PSPACE complete [9]:

- Impartial Game Positive 11-SAT (11-CNF)
- Impartial Game Positive 11-DNF SAT
- Partizan Game CNF SAT
- Impartial/Partizan Avoid Positive 2-DNF SAT
- Impartial/Partizan Seek Positive 3-DNF SAT
- Impartial/Partizan Avoid Positive CNF SAT
- Impartial/Partizan Seek Positive CNF SAT

4.2 Kayles

Kayles is a 2 player version of independent set, with two major forms.

Impartial Node Kayles allows players to alternate turns. Each player may choose any node in the graph to add to the graph's independent set. The player who cannot move on a given turn loses.

Partizan Bipartite Node Kayles is similar to the version of Kayles above. However, the bipartite restriction refers to the fact that the white nodes and the black nodes form a bipartition, rather than the graph itself being bipartite.

Both versions of this game are PSPACE-complete [9].

4.3 Geography

Geography is a generalization of the word game, where players alternate stating words that begin with the same letter as the last letter of the previous word (Antartica, Africa, Alabama, ... is an example of a set of moves).

Geography is played on a directed graph with a token placed on a start node. During each turn, a player moves a token along a directed edge. There are two main variants of Geography.

The first is Node Geography, where players may not revisit nodes. Node Geography on a directed graph is PSPACE-complete [5], while Node Geography on an undirected graph is in P [2].

The other variant is Edge Geography, in which a player may not revisit an edge. Edge Geography is PSPACE-complete on directed graphs [9] and undirected graphs, but in P for bipartite graphs [2].

5 Reversi/Othello

Reversi (also known as Othello) is a game played with black and white pieces on a grid-board. Each move consists of playing a piece so that a straight line (vertical, diagonal, or horizontal) of

opponent pieces becomes sandwiched by two pieces of your color. The opponent pieces are then flipped to your color. If a player has no moves, he must pass. The goal of the game is to maximize the number of pieces of your color. Our decision problem is whether black has a winning strategy.

Because each move “consumes” a square on the board, there are a bounded number of moves so Reversi is in PSPACE. We can show that Reversi is PSPACE-complete by reduction from directed, bipartite, max-degree 3 node geography [4].

In our construction, black makes moves around the board flipping long lines of white pieces, which are edges, and there are places where white can choose which path black takes (as well as where black can choose a path). We have a large region of white on the bottom, with two columns of white pieces with gaps along the right side. If black is able to enter the white columns, he can play into the bottom right corner, taking the right edge, and then take the entire bottom region to win.

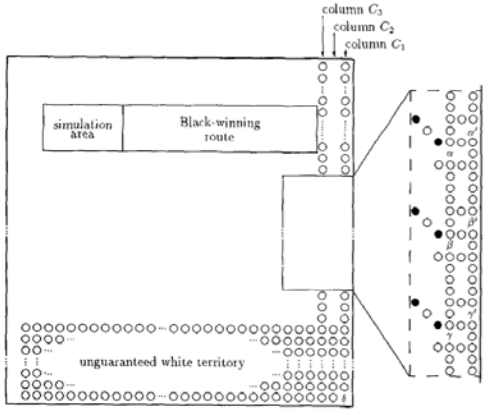


Figure 16: The overall configuration for our reduction. If black can win if he can place his piece in the bottom right corner, where it will be unflippable, and flip all of white’s territory.

We can then create Shift gadgets and Turn gadgets (which also represent degree 2 vertices) to be able to create arbitrary edges. Threat lines ensure that black and white don’t deviate from the line of play.

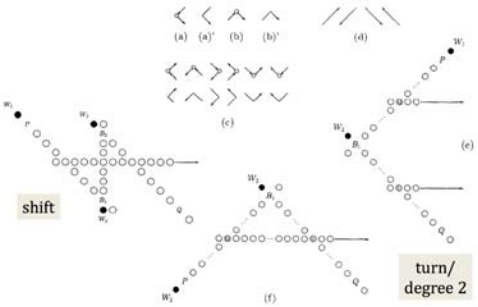


Figure 17: Shift and Turn (degree-2) gadgets. Note that because of the way the victory gadget is oriented with the win column on the right, the turn gadget differs depending on the orientation of the gadget.

Because the graph is bipartite, we know which player acts at or visits which node. Thus, we can

create gadgets for the 4 kinds of degree three node: a vertex with 1 edge out that white visits, a vertex with 1 edge out that black visits, a vertex with 2 edges out that white makes a choice at, and a vertex with 2 edges out that black makes a choice at.

We construct the gadgets for vertices with 1 edge out so that visiting the gadget twice causes results in a loss. For white, this means that if white visits a vertex twice, then black can activate a threat line, and win the game. For black, if a vertex is visited twice, white can block the path, and black will have no moves left, losing the game.

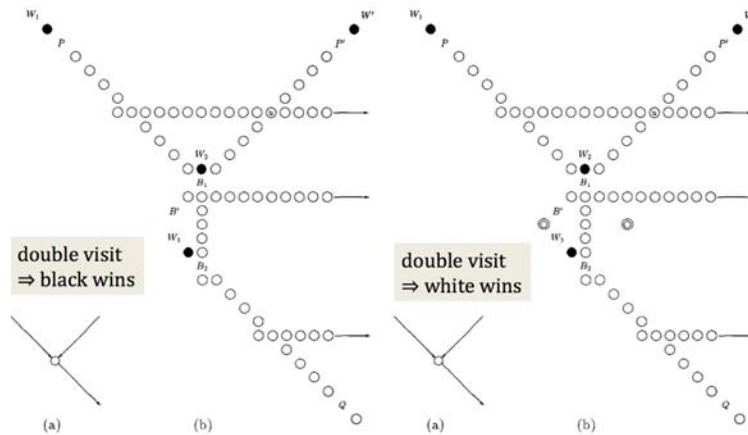


Figure 18: Degree-3 vertices with two edges going in. If either white or black visits the vertex twice, the threat lines going to the right ensure that the player visiting loses.

We can also construct the gadgets for choosing.

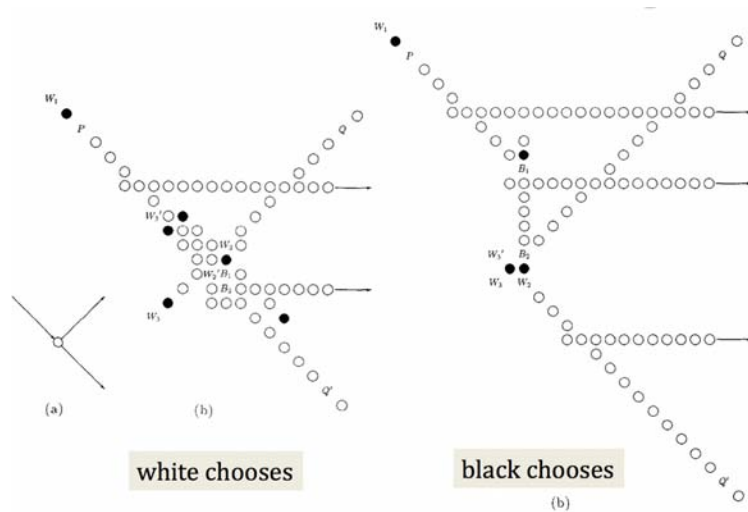


Figure 19: Degree-3 vertices with one edge going in. Afterwards, either white or black has a choice on which node to visit.

This completes the reduction from directed, bipartite, max-degree 3 node geography.

References

- [1] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for your Mathematical Plays*, volume 2. Academic Press, New York, 1982.
- [2] Aviezri S. Fraenkel, Edward R. Scheinerman, and Daniel Ullman. Undirected edge geography. *Theor. Comput. Sci.*, 112(2):371–381, 1993.
- [3] Robert A. Hearn and Erik D. Demaine. *Game, Puzzles, and Computation*. A K Peters/CRC Press, 1st edition, July 2009.
- [4] Shigeki Iwata and Takumi Kasai. The othello game on an $n \times n$ board is pspace-complete. *Theoretical Computer Science*, 123(2):329–340, January 1994.
- [5] David Lichtenstein and Michael Sipser. Go is polynomial-space hard. *J. ACM*, 27(2):393–401, April 1980.
- [6] Marvin Minsky. *Computation: Finite and Infinite Machines*, pages 255–258. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1st edition, 1967.
- [7] Paul Rendell. Turing universality of the game of life. In Andrew Adamatzky, editor, *Collision-Based Computing*, pages 513–539. Springer London, 2002.
- [8] Thomas J. Schaefer. The complexity of satisfiability problems. *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226, 1978.
- [9] Thomas J. Schaefer. On the complexity of some two-person perfect information games. *J. Comput. Syst. Sci.*, 16:185–225, 1978.

MIT OpenCourseWare
<http://ocw.mit.edu>

6.890 Algorithmic Lower Bounds: Fun with Hardness Proofs
Fall 2014

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.