6.890: Algorithmic Lower Bounds: Fun With Hardness Proofs Fall 2014

Lecture 6 Scribe Notes

Prof. Erik Demaine

Overview

Today is the last lecture about pure 3-SAT, and we'll be focusing in particular on circuit SAT today. Before that we will compare and contrast some of the aspects of the two styles of 3-SAT reduction proofs we've seen.

Dual-rail logic vs. Binary logic

Dual-rail logic

The idea of Dual-rail logic is to use two "semiwires" per variable. Each (semi)wire either holds the value TRUE or doesn't. The choice of which wire holds the value TRUE corresponds with the value of the variable. This is in contrast to Binary logic (below) which represents each variable as a single wire which can hold either TRUE or FALSE values.

As an example, consider any of the Nintendo proofs from previous classes. In those proofs, each variable has a separate rail/line for the true case and a seperate line for the false case.

In addition to a *(semi)wire gadget*, this style of proof requires the use of a *variable gadget*, which forces at most one of two "semiwires" to be on.

Binary logic

The idea of Binary logic is to use a wire which can be solved in exactly one of two ways. The choice of which solution is used encodes a value of TRUE or FALSE.

This style of proof requires the following gadgets

- A *wire gadget* has 2 possible solutions. The choice of which to use encodes a binary value.
- A *split gadget* guarantees that the two (or more) output wires hold the same value as the input wire.
- A *NOT gadget* guarantees that the input and output wires have opposite values. This is only necessary if negation is required: in the non-monotone case.
- A *terminalor gadget* leaves the wire free to choose true or false. For example, in the origami proof, the edge of the paper serves as a sort of terminator gadget. Sometimes this gadget is unneccessary.

Both cases (binary and dual-rail)

The two styles of proof have a lot in common. The following is a list of gadgets that are necessary in both:

- A *turn gadget* is often needed when embedding in two dimensions to send wires in the correct directions.
- A *crossover gadget* allow wires to cross without interacting. This is not necessary if reducing from a planar version of SAT.
- A *shift gadget* can be used to fix parity/modulo errors such as what we saw in the Planar Vertex-Disjoint Paths problem.

Circuit SAT

Reduction from Circuit SAT is an alternative to the above two styles of proof. Circuit SAT asks the question of whether a given logic circuit has an assignment for its inputs that causes the output to be true.

In addition to wires, splitters, and gates, a reduction from Circuit SAT requires a TRUE terminator gadget in order to constrain the output wire to be true.

Planar Circuit SAT

Planar circuit SAT (careful: this is not widely used terminology) is the following problem:

You are given a planar directed acyclic graph. Each node in the graph is either a source (of indegree zero), a NAND gate (of indegree 2 and any outdegree), or a sink (outdegree zero). There must be exactly one sink and its indegree must be zero. The question is whether the edges (aka wires) can be colored with the two colors TRUE and FALSE such that the following hold:

- The value of the edge going to the sink is TRUE.
- The value of any edge out of a NAND gate is the NAND of the truth values of the two edges going into the gate.
- The value of an edge out of a sink can be anything.

This problem is NP-complete.

In fact, the problem is also NP-complete if we replace the word NAND with the word NOR everywhere. This is because both NAND and NOR are universal gates.

Puzzles

Next we will discuss the hardness of several puzzles.

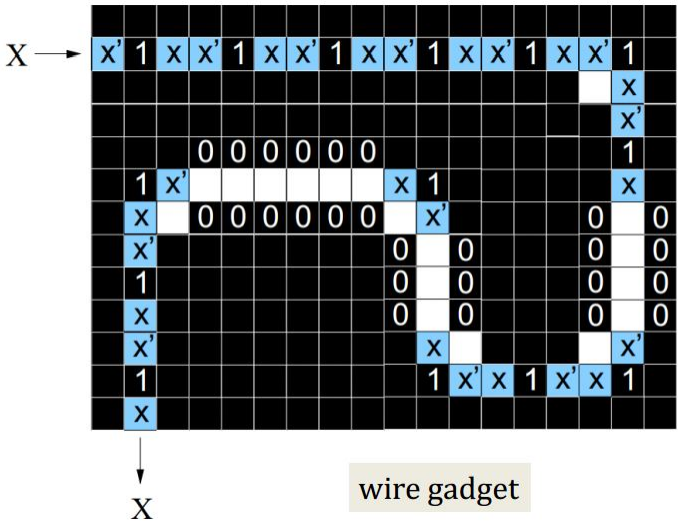
Akari / Light UP [Nikoli 2001]

An instance of the puzzle Akari consists of a grid of white and black squares where some black squares have a number between 0 and 4 written on teh. The goal of the puzzle is to place lights on the white squares of the board so that the following rules are satisfied:

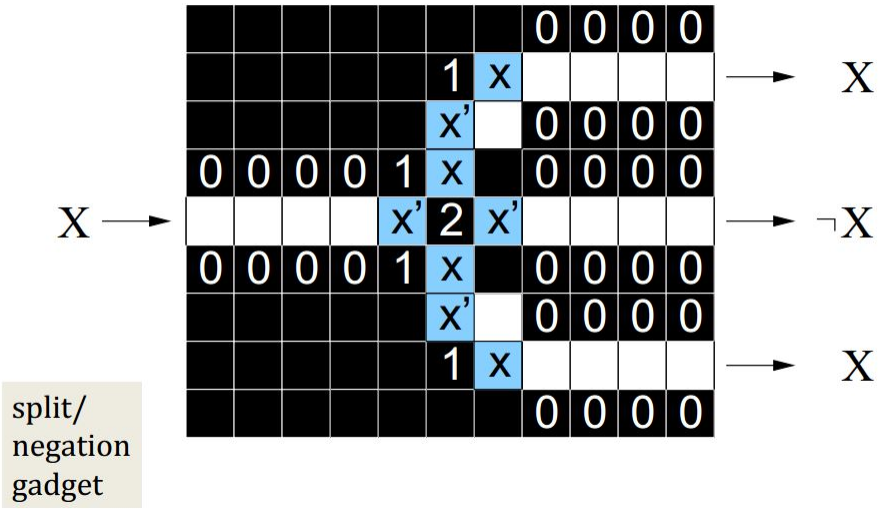
- Each black square which has a number written on it must have that number of lights in a horizontally or vertically adjacent square.
- Each white square should be lit up by exactly one light. A light lights up a square if both the light and the square are in the same row or column and if there are no black squares between them

It has been proven that the problem of deciding whether a given board has a solution is NP-comple [McPhail 2005].

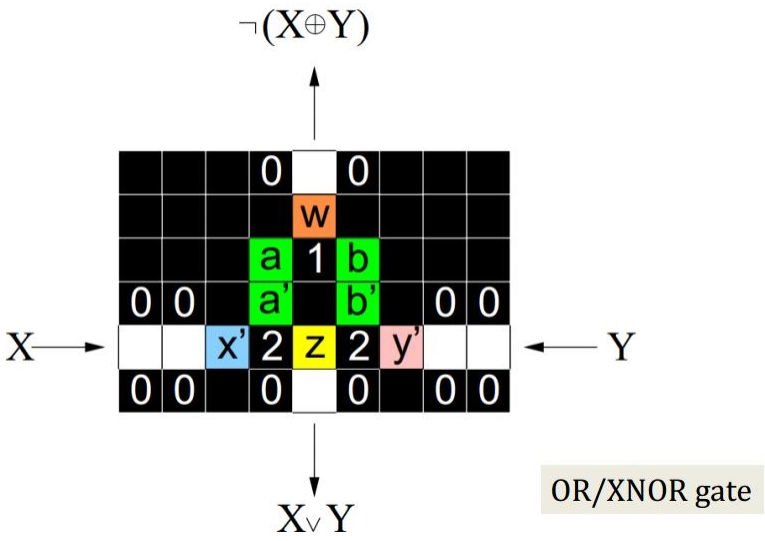
Like most 3-SAT reduction NP-hardness proofs, this proof starts with the wire gadget. A basic wire gadget consists of a repeating sequence of a black square with a 1 followed by two white squares. This forces exactly one of the two white squares to have a light in a way such that the choice of using the white square on the right or on the left propogates down the wire. Alternatively, by lining a corridor with 0's, one can have a different wire because the light must be at one of the ends of the corridor. Making turns is also easy. All of these gadgets are depicted below.



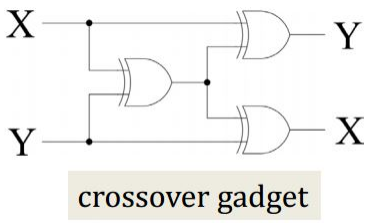
The negation/split gadget below outputs a single negated copy and two regular copies of the incoming wire. By using a terminator gadget, you can use this gadget as either the split gadget or the not gadget.



Similarly, the next gadget behaves as both an OR gate and an XNOR gate. Showing that this is true requires simply going through all the cases.



The following circuit diagram shows that a planar crossover gadget can always be built from three XOR gates. Since a planar XOR can be built from NOR and XNOR, we automatically have a crossover gadget.

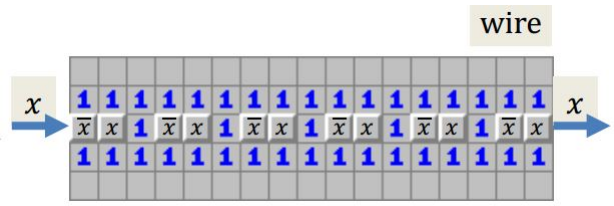


The above gadgets, together with a terminator gadget (which requires simply placing a 0 or 1 at the end of a wire to set its value), are sufficient to prove that Akari is NP-hard.

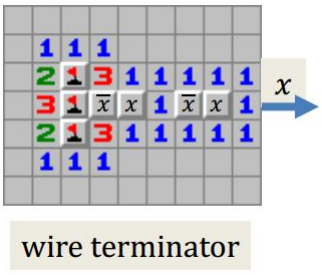
Minesweeper Consistency is NP-complete [Kaye 2000]

The Minesweeper Consistency problem asks whether a given minesweeper board is possible. The motivation for this problem is that if we could solve the problem efficiently then placing a bomb in a location and then checking whether the board is consistent allows a player to check whether that location is safe to click. One of the problems with this approach to playing minesweeper is that there might not be any safe moves. As a result, this isn't the best problem to pose, but it is the first one to be proved hard.

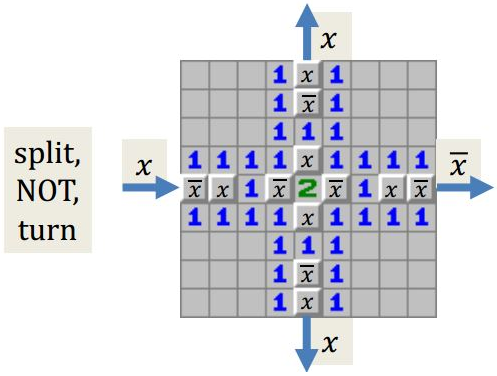
The wire gadget for minesweeper looks a lot like the Akari wire. The bombs must all be placed in the right square or the left of the pairs of unspecified squares.



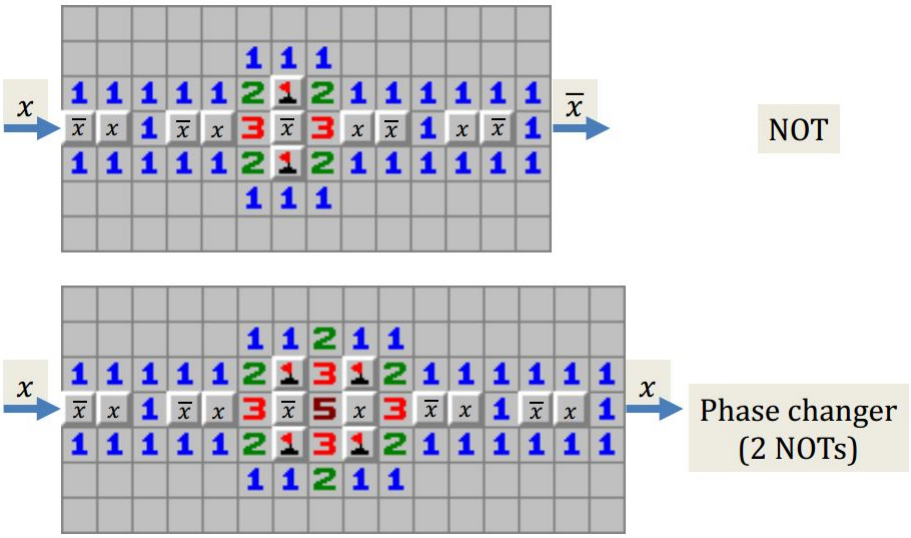
The terminator gadget for minesweeper is more complicated than the Akari terminator because simply stopping a wire would require specifying that the last square has a bomb, which would not leave the wire unspecified.



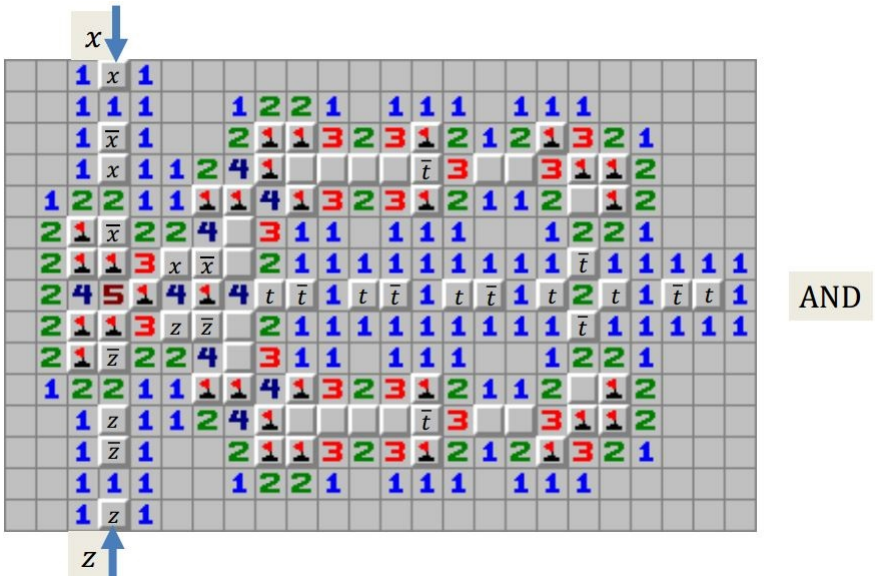
The following gadget is a split, NOT, and turn gadget all in one. Each individual gadget can be extracted from this design by adding terminators to the unnecessary wires.



The more simple NOT gate below can be used twice in order to fix the "mod-3-parity" issue (that wires are always a multiple of 3 length).



The AND gadget looks very complicated, but all one has to do to check it is check all 4 cases.



Together, the AND and NOT gates create a NAND, which is universal. In addition, planar XOR can be built from 3 NANDs, so we get a crossover gadget for free (remember, XOR is enough to build a crossover).

Building a true terminator is easy (by just ending the wire with a 1), so the above gadgets are enough to prove NP-hardness.

Note again that the question we asked isn't really the right one. Also note that the number of bombs used in each gadget varies according to the truth values of the variables. In actual minesweeper, the number of bombs is specified, so that is another problem.

Minesweeper is coNP-complete [Scott, Stege, van Rooij 2011]

Here, when we talk about Minesweeper, we are talking about the Minesweeper Inference problem: is it possible to locate all of the bombs given a minesweeper board.

To prove that Minesweeper is coNP-hard we reduce from Circuit UNSAT (a known coNP-complete problem).

Circuit UNSAT asks whether for a given formula f the following is true:

$$\neg\exists x_1 \exists x_2 \cdots \exists x_n : f(x_1, \ldots, x_n) = 1$$

$$\equiv \forall x_1 : \forall x_2 : \cdots \forall x_n : \neg f(x_1, \ldots, x_n) = 1$$

coNP is the class of problems for which every non-solvable instance has a polynomial length certificate for non-existance of a solution. Contrast this with NP, which contains all problems for which every solvable instance has a polynomial length certificate for the existance of the solution.

coNP and NP are about the same level of "hard". In fact, it turns out that if P = NP or if we allow multicall reductions, then NP = coNP.

In this reduction, we make sure that all of our gadgets have the same number of bombs regardless of variable assignment.

The wire gadget stays the same, but many of the other gadgets get much more cluttered. This time we use an OR gate, which together with NOT gives the universal gate NOR. We can construct NAND out of a planar arrangement of NORs, so again we get a free crossover.

The final output of the circuit is connected to a terminal gadget that leaves it undetermined rather then setting the output to true as we would in a SAT reduction. The reason is as follows: if the UNSAT instance is always false then there will always be a bomb in the end of the circuit; if some assignment exists that makes the output true then there exists some solution to the minesweeper instance that has no bomb there. Thus, since it is hard to determine whether a formula is in UNSAT it is also hard to determine whether a bomb must exist in a particular location on the minesweeper board.

Candy Crush is NP-complete [Walsh 2014]

Candy Crush is a game where you are given a grid of colors (colored candies). The player may swap two adjacent candies as a move provided a three-in-a-row of the same color is formed. The physics of the game is that whenever you have such a three-in-a-row of the same colored candies in the same row or same column, they disappear, possibly causing a chain reaction. In this proof, we'll assume that if there are multiple 3 in a rows at once, disappearing resolves from bottom up.

The problem of whether it is possible to get some minimum number of points in a given number of clicks is NP-hard by reduction from SAT.

Most of the candies are positioned in a medium of four colors in an alternating pattern. No moves are possible in this medium, but if some pieces are replaced by a fifth color (i.e. purple) then the new color might allow moves.

The variable gadget consists of three columns and can be activated to make either the right or middle column fall 3 squares. This is meant to trigger further activity above the variable gadget.

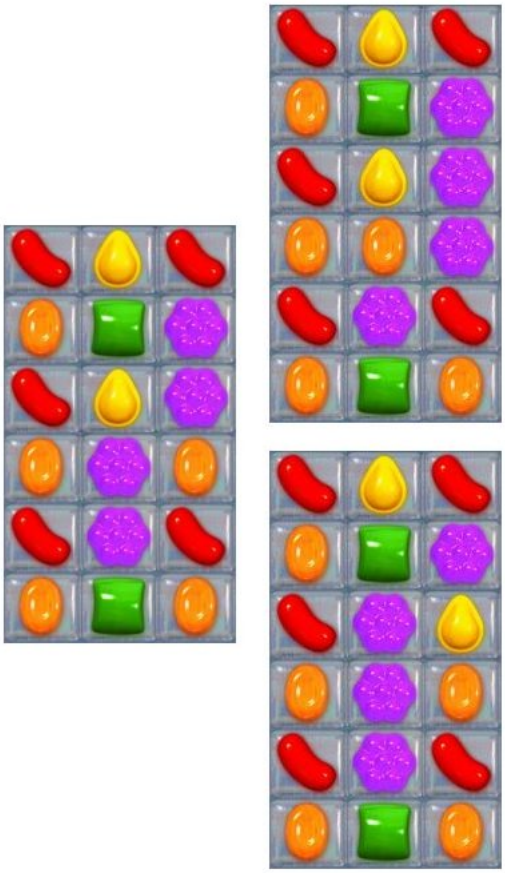
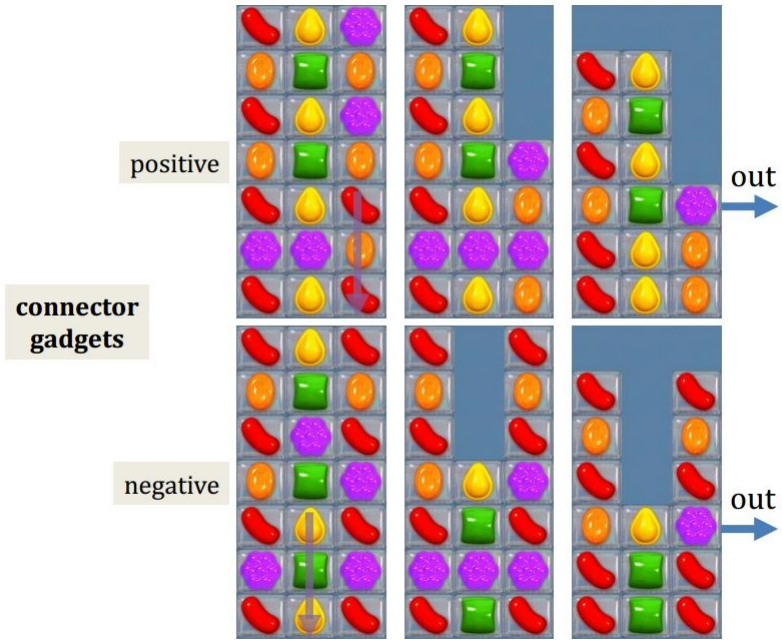
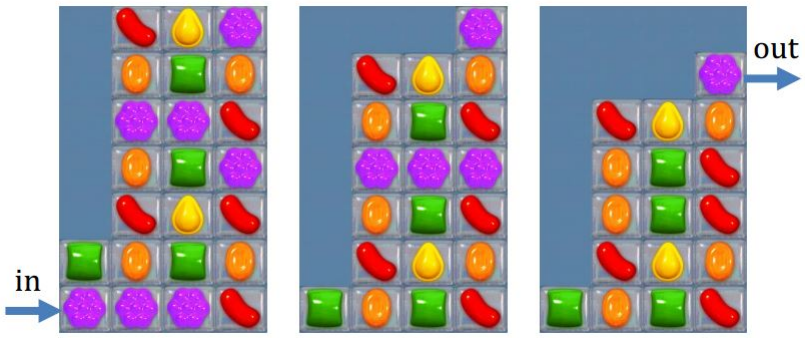


Figure 1: The variable gadget on the left can be activated one of two ways (on the right)

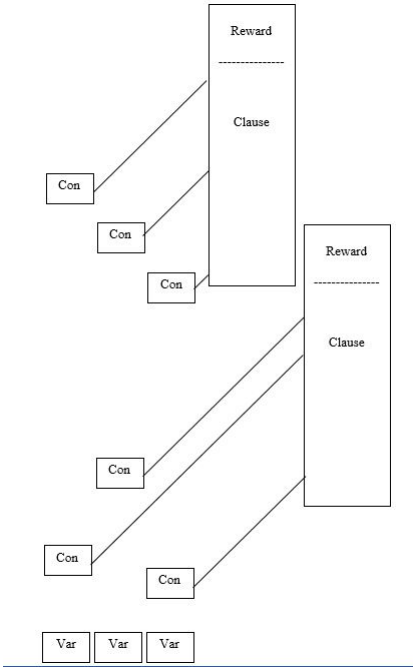
Above the variable gadget is a connector gadget which transforms the downward motion caused by the variable's activation into lateral motion in a wire (which will be introduced next).



The connector transforms downward motion by three squares in one column into downward motion by one square in a column further to the right. The wire gadget transforms downward motion by one square in one column into downward motion by one square in a column further to the right:



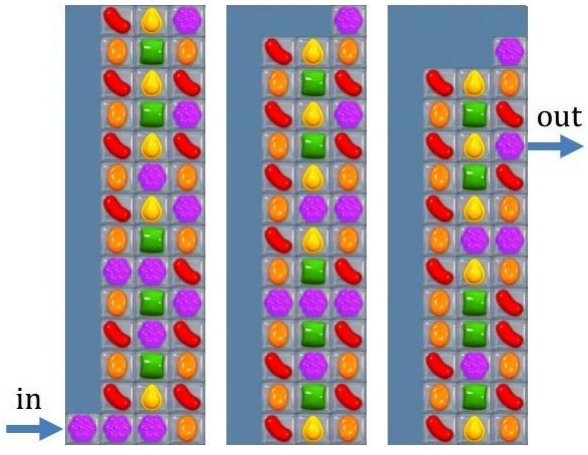
The overall idea is to have the gadgets arranged in the following pattern:



The rewards in the scheme are huge regions which grant "a million" points if the clause gadget is activated. To win the game, the player wants as many points as possible, which is only possible by satisfying all the clauses.

There is a small problem with the above diagram: some of the wires go over a variable. This means that if the variable is activated before the wire is triggered, the wire is no longer usable.

To solve this problem, we use the following improved wire gadget, which remains usable in all of the cases: if the right column falls three squares, the middle column falls three squares, or neither of these things happen.



This is sufficient to complete the proof.

Note that a player can directly activate a wire, but we can argue that the player won't ever get as many points from a single clause as from setting a variable. We can make it even better by

making a "million" copies of the clause, so you get a million times a million points from activating a variable that satisfies a clause, which is way more than you can get by manually setting any one wire. Remember, the player has a limited number of moves, which is why they can't set all the clauses directly.

Bejeweled, Candy Crush, ... are NP-Complete [Guala, Leucci, Natale 2014]

A whole general class of 3-in-a-row puzzles was proved hard using a reduction from 1-in-3-SAT. The details are complicated, but the reduction was actually implemented, and can be played at http://bejeweled.isnphard.com/. Unfortunately, even just the formula $(x_1) \wedge (x_1 \vee x_2)$ requires approximately 6000 rows in the corresponding game instance.

6.890 Algorithmic Lower Bounds: Fun with Hardness Proofs

Fall 2014