Circuit Satisfiability Problem, CSAT
CS351

The satisfiability problem was proved by Stephen Cook in the early 70's to be the first NP Complete problem. Here we will give the basic argument for a related problem, that of the circuit satisfiability problem. The actual proof given by Cook involves quite a few more details. Recall that to show something is NP Complete means that the problem is in NP, and that all other problems in NP can be reduced to the NP Complete problem in polynomial time.

First, we need the concept of a **certificate**. A certificate is just a proposed solution to a problem. A certificate is used by a verification algorithm, V, where V is used to verify if the certificate is valid or not on input x (i.e. it is used to test if a problem is in NP).

Algorithm V verifies an input string x if there exists a certificate y such that V(x,y) = true. The language verified by V can be stated as:

$$L = \{ x \mid V(x,y) = 1 \text{ for some string } y\}$$

In these terms, x is the problem statement (e.g. for Hamilton Cycle, it is the graph and all the edges). Y is the certificate, or the proposed solution (e.g. for Hamilton Cycle, a list of all vertices in the cycle). We measure the time of a verifier only in terms of the length of x, so a polynomial time verifier runs in polynomial time in the length of x. If a language has a polynomial time verifier, then by definition it is in NP. For polynomial time verifiers, the length of the certificate must be polynomial bounded to the length of x.

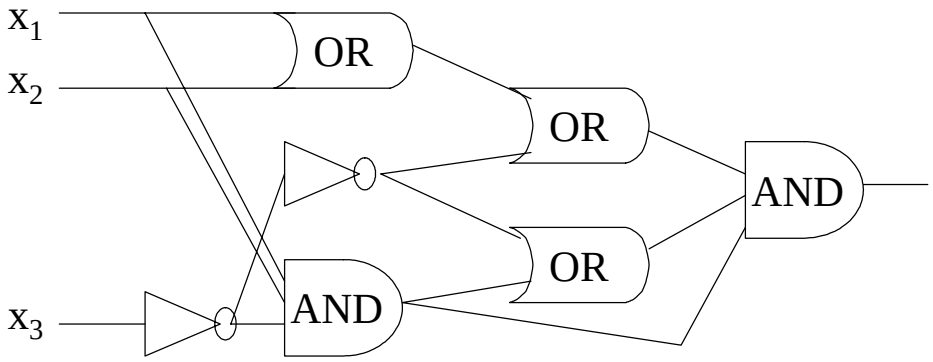
Circuit Satisfiability

The CSAT problem is very similar to the Satisfiability problem, but uses actual digital circuits instead of Boolean expressions.

The CSAT problem is:

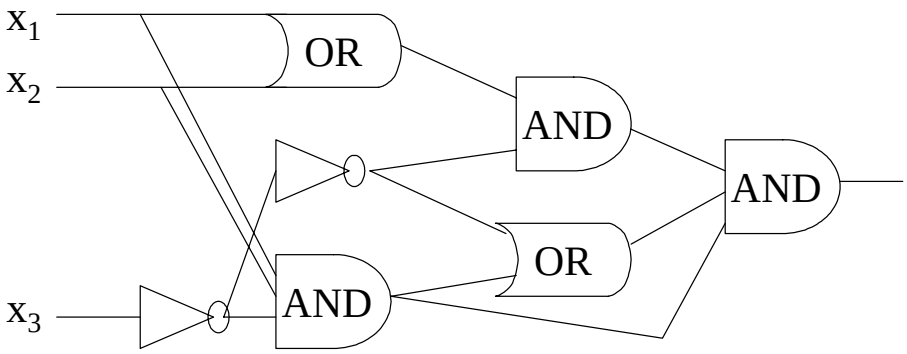
> Given a Boolean combinational circuit composed of AND, OR, and NOT gates, is it satisfiable?

i.e. is there a set of inputs that makes the output 1? Consider the example below:



In the above circuit, can you find values for X_1, X_2, and X_3 that makes the output 1? Apparently we may need to simply try all the (exponential) possibilities until we find a satisfying assignment. In this case the assignment 1,1,0 makes the output 1.

However, not all circuits are satisfiable. If we change the second OR to an AND the following is not satisfiable:



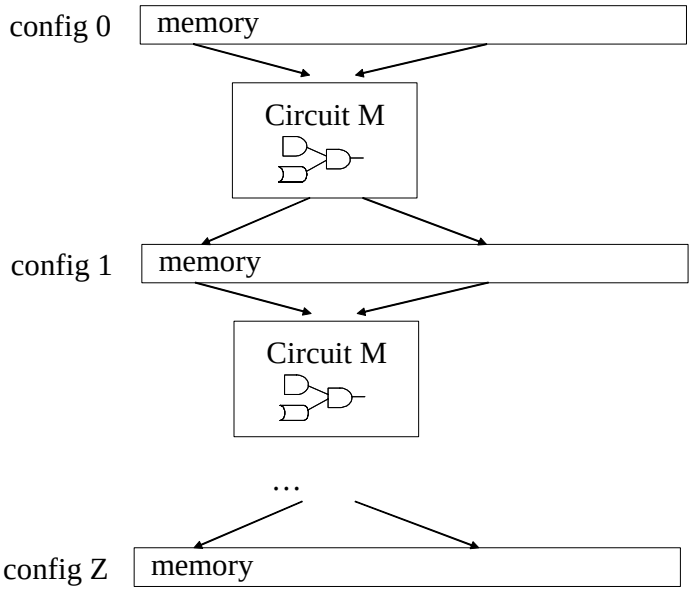
Our claim is that CSAT is NP-Complete. To make this claim, we must first show that CSAT belongs to the class NP.

This part is easy to show. Recall the verification algorithm, V(x,y) where x is the problem specification and y is a certificate. Our verification algorithm is to construct the actual circuit out of the input specifications, x. Then, y is a proposed solution. Put the values of y into our constructed circuit and then simulate it. If the output is 1, then the circuit is satisfiable, otherwise 0 is output. This entire operation can be completed in time polynomial to the input, x, thus CSAT is in NP.

Second, we must show that every language in NP is polynomial-time reducible to CSAT. Our proof is based on the workings of an actual computer (and thus is translatable to the workings of a Turing Machine). This is much harder to prove, and here we only give a sketch of the formal proof.

Consider a computer program. The program operates as a sequence of instructions stored in memory. Furthermore, data is stored in memory. We have a program counter and other registers to operate on data and store the current instruction. Let's just consider all registers plus RAM to be "memory". At any point in the execution of a program, the entire state of the computation is represented in the computer's memory. A particular state of computer memory is a configuration. The execution of an instruction may be viewed as mapping one configuration to another. In terms of traversing states, a program is traversing a sequence of configurations.

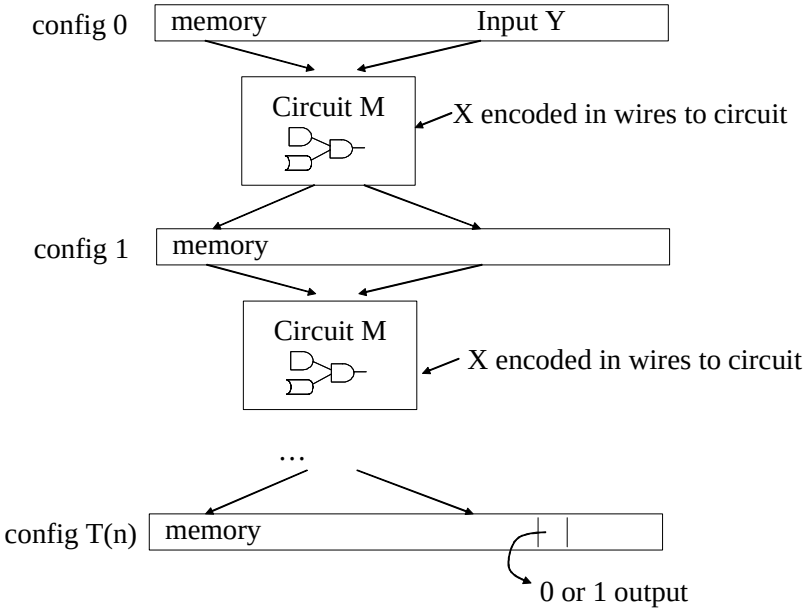
This mapping from one configuration to another can be accomplished by a combinational circuit (in fact, this is what is done by the computer). Call this combinatorial circuit, M. The execution of a program taking Z+1 steps can then be viewed as the following:



Now, we are ready to show that every language in NP is polynomial-time reducible to CSAT.

Let L be any language in NP. By definition, L has a verification algorithm, V(x,y), that runs in polynomial time. This means that if the input x is of length n, then there is a constant k such that the runtime of V, T(V), is $O(n^k)$. Similarly, the length of the certificate, y, must also be $O(n^k)$.

Since V consists of a polynomial number of steps, then in polynomial time we can construct a single combinational circuit that computes all configurations produced by an initial configuration. The input of x can be hardcoded to circuit M in terms of certain wires (e.g. for Hamilton Circuit, we could have wires encoding the nodes and edges in the graph). This is because x never changes for each step of the configurations. This leaves y as input values to the circuit. In this case, we don't know what y is – we want to find values of y that satisfy the circuit (and therefore solve the original NP problem). The picture is as follows:



The last configuration is T(n), the runtime of V. We use some location in memory to get the ultimate output for the satisfiability of the circuit.

If we take a step back, what have we just created? It is simply a big version of a CSAT problem! The input to the problem is configuration 0, and the output is configuration T(n). In between is a big combinatorial circuit. We now have reduced the problem in NP to the CSAT problem. In this instance of CSAT, the input is certificate Y and we don't know what the certificate Y is (i.e. we don't know a proposed solution). If we solve this CSAT problem, then we will have determined if an input Y exists or not. If this input Y exists, then it is a solution to the original problem in NP. Thus, if CSAT is satisfiable, then Y solves the NP problem.

In the other direction, suppose that some certificate exists. When we feed this certificate into the circuit, it will produce an output of 1. Thus, if the original problem is solvable, this instance of CSAT is also satisfiable.

To complete the proof, we must show that the circuit can be constructed in polynomial time – i.e. the reduction is polynomial. Since the verification algorithm runs in polynomial time, there are only a polynomial number of configurations. This means we are hooking together some polynomial number of circuits M. We can make an argument that M can be constructed in size polynomial to the length of a configuration – M is essentially the construction of the computer hardware for a simple CPU and memory system. This makes the overall construction time polynomial.

Based on the two properties of CSAT, we conclude that CSAT is therefore at least as hard as any language in NP and since it is in NP, it is NP Complete.