COMPSCI 330: Design and Analysis of Algorithms **December 5, 2017**

Lecture 24 : Even more reductions

Lecturer: Yu Cheng *Scribe: Will Wang*

1 Overview

Last two lectures, we showed the technique of reduction with some examples. In this lecture, we first review the basic concepts and then show three more examples on reduction: Vertex 3-Coloring, Hamiltonian Cycle and Super Mario.

1.1 Polynomial time reductions

Here we review some basic concepts about reduction.

Reduce A to B: a polynomial time algorithm that maps instances of A to instances of problem B, such that the answers are the same.

A$\leq_p$B: B is at least as hard as A. If you can solve B (in poly time) then you can solve A.

Gadget-Based Reductions: Given instances of A, we need output instances of B. What we do is to first build gadgets for pieces of A and put the pieces together.

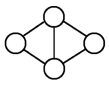
2 Vertex 3-Coloring

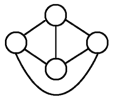
Consider the following Vertex 3-Coloring problem:

Input:a graph

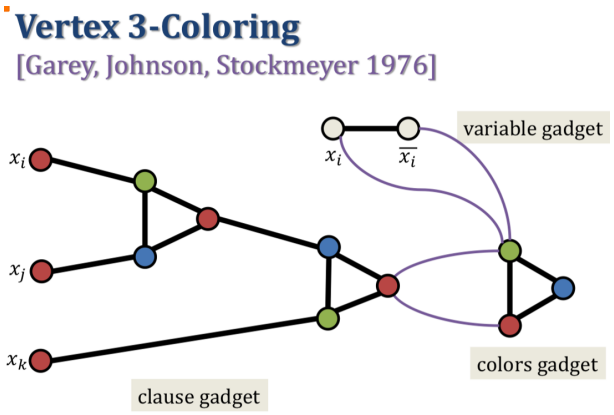
Output: color each vertex using 1 of the 3 colors, so that adjacent vertices do not get the same color.

We here show an reduction from 3SAT to Vertex 3-Coloring.

Satisfiable formula $\Leftrightarrow$ 

Unsatisfiable formula $\Leftrightarrow$ 

We show reduction by building following gadgets. For each clause, we can build such a graph on which we run Vertex-Coloring.



One direction: If the 3SAT formula is satisfiable, then our graph is 3-colorable.

Another direction: If the 3SAT formula is UN-satisfiable, then our graph is NOT 3-colorable.

Therefore, we show 3-Coloring is NP-Complete by reduction. (Because 3-Coloring is also in NP)

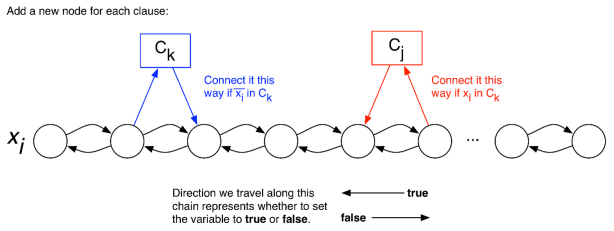
3 Hamiltonian Cycle

Now, consider the following Hamiltonian Cycle problem:

Input: A (directed) graph.

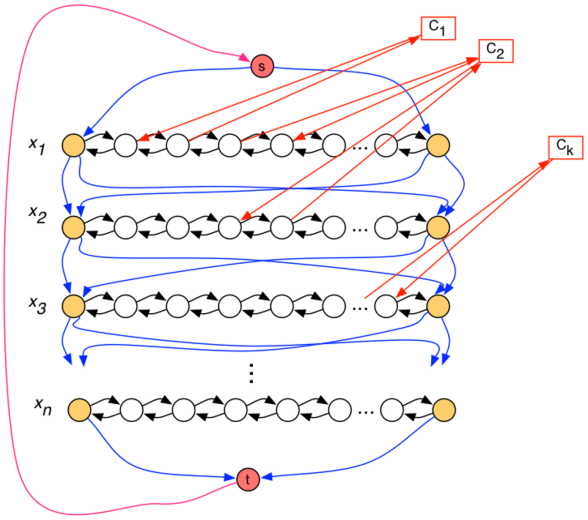
Solution: A cycle visiting every vertex exactly once.

To reduce 3-SAT to Hamiltonian Cycle, we design the gadget as below:

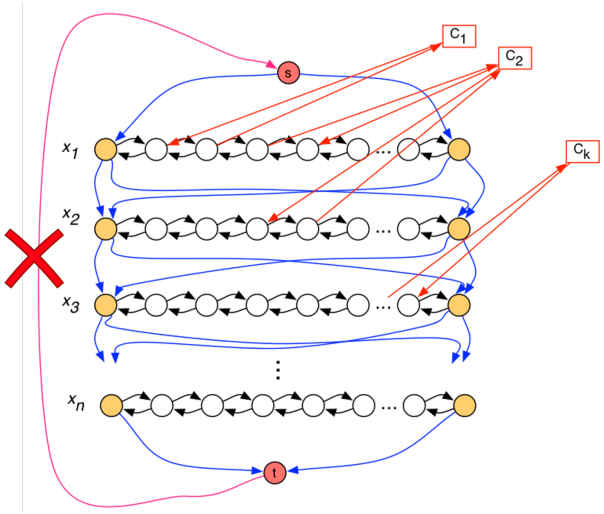


A Hamiltonian path encodes a truth assignment for the variables (depending on which direction each chain is traversed).

For there to be a Hamiltonian cycle, we have to visit every clause node. We can only visit a clause if we satisfy it (by setting one of its terms to true). Hence, if there is a Hamiltonian cycle, there is a satisfying assignment.



And naturally, if there is not a Hamiltonian cycle, there is not a satisfying assignment.

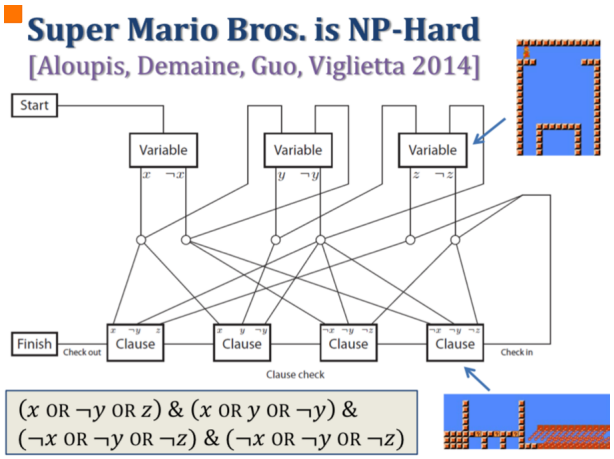


4 Super Mario

In the end, we can actually reduce 3-SAT to Super Mario and show it is NP-complete. The following descriptions are from the paper "Classic Nintendo games are (computationally) hard." Theoretical Computer Science 586 (2015): 135-160 by Aloupis, Greg, et al.

The framework reduces from the classic NP-complete problem 3-SAT: decide whether a 3-CNF Boolean formula can be made true by setting the variables appropriately. The players character starts at the position labeled Start, then proceeds to the Variable gadgets. Each Variable gadget forces the player to make an exclusive choice of true (x) or false (x) value for a variable in the formula. Either choice enables the player to follow paths leading to Clause gadgets, corresponding to the clauses containing that literal (x or x). These paths may cross each other, but Crossover gadgets prevent the player from switching between crossing paths. By visiting a Clause gadget, the player can unlock the clause (a permanent state change), but cannot reach any of the other paths connecting to the Clause gadget. Finally, after traversing through all the Variable gadgets, the player must traverse a long check path, which passes through each Clause gadget, to reach the

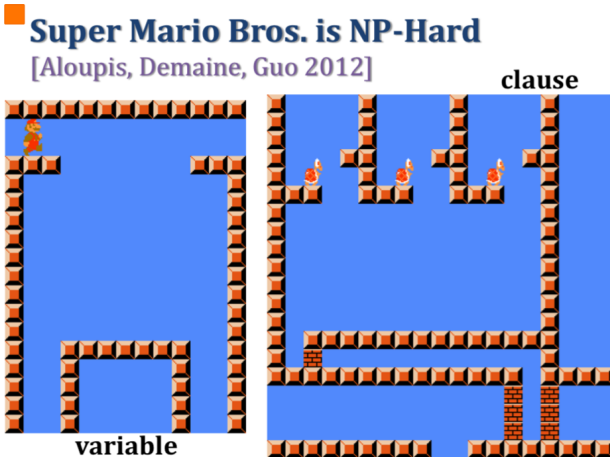
Finish position. The player can get through the check path if and only if each clause has been unlocked by some literal. Therefore, it suffices to implement Start, Variable, Clause, Finish, and Crossover gadgets to prove NP-hardness of each platform game.



The specic properties the gadgets must satisfy are the following:

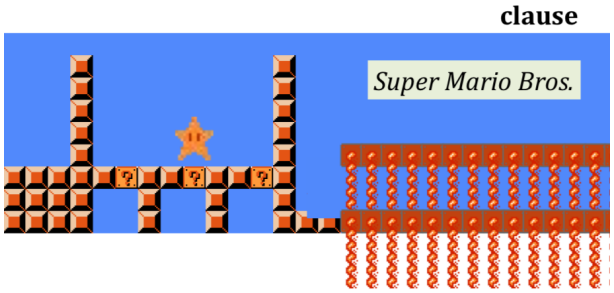
Start and Finish: The Start and Finish gadgets contain the starting point and goal for the character, respectively. In most of our reductions, these gadgets are trivial, but in some cases we need the player to be in a certain state throughout the construction, which we can force by making the Finish accessible only if the player is in the desired state, and the desired state may be entered at the Start. For example, in the case of Super Mario Bros., we need Mario to be big throughout the stage, so we put a Super Mushroom at the start and a brick at the Finish, which can be broken through only if Mario is big.

Variable: Each Variable gadget must force the player to choose one of two paths, corresponding to the variable xi or its negation xi being chosen as the satised literal, such that once a path is taken, the other path can never be traversed. Each Variable gadget must be accessible from and only from the previous Variable gadget, independent of the choice made in the previous gadget, in such a way that entering from one literal does not allow traversal back into the negation of the literal.



Clause and Check: Each Clause gadget must be accessible from (and initially, only from) the literal paths corresponding to the literals appearing in the clause in the original Boolean formula.

In addition, when the player visits a Clause gadget in this way, they may perform some action that unlocks the gadget. The Check path traverses every Clause gadget in sequence, and the player may pass through each Clause gadget via the Check path if and only if the Clause gadget is unlocked. Thus the Check path can be fully traversed only if all the Variable gadgets have been visited from literal paths. If the player traverses the entire Check path, they may access the Finish gadget.



Crossover: The Crossover gadget must allow traversal via two passages that cross each other, in such a way that there is no leakage between the two.