Notes, 9(b)

ECE 606

Why care about non-determinism?

Because there is a class of decision problems that demonstrate a peculiar feature that we can naturally associate with non-determinism. A number of such problems arise in practice in various fields.

The feature is: it may not necessarily be easy to solve a given instance of the problem. However, given a proposed solution, it is easy to check that it is indeed a solution.

Example: given a Sudoku puzzle, does it have a solution?

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Note: Sudoku as we know it is constant-size: always 9×9. But we can think of generalized Sudoku, which comprises an $n^2 \times n^2$ grid. That version also has the feature that it is easy to check whether a claimed solution is indeed correct.

This feature of being able to easily check the goodness of a solution corresponds exactly with an efficient non-deterministic algorithm.

= polynomial-time

Another feature these problems possess: if an efficient algorithm exists for the decision version of the problem, then an efficient version exists for the construction or optimization version of the problem.

For example, for Sudoku, assume that there exists an *oracle* for the decision version of the problem: given a Sudoku puzzle, does it have a solution?

Then, we have a simple algorithm for constructing a solution, if the oracle says "yes" for the puzzle. Try each number, in turn, for the first blank slot, each time querying the oracle. As soon as it says "yes" for a number in the slot, move on to the next slot and repeat this process. This procedure involves at most n^2 attempts for each for the n^4 slots, for a total running-time of n^6, which is polynomial in the input-size of n^4.

There are a number of problems that exhibit the above two features. Some examples:

SORTEDARRAY: given an array of n items, is it sorted?

VERTEXCOVER: given an undirected graph G and an integer k, does G has a vertex cover of size k?

HAMPATH: given an undirected graph G, does it have a simple path of all its vertices? (Such a path is called a "Hamiltonian path.")

HAMCYCLE: given an undirected graph G, does it have a simple cycle of all its vertices? (Such a cycle is called a "Hamiltonian cycle.")

SHORTSIMPLEPATH: given an undirected graph G, two vertices in it a, b, and an integer k, does there exist a simple path $a \rightsquigarrow b$ of at most k edges?

LONGSIMPLEPATH: given an undirected graph G, two vertices in it a, b, and an integer k, does there exist a simple path $a \rightsquigarrow b$ of at least k edges?

SUBSETSUM: given a set of integers, does some subset comprise members that add up to 0?

SETCOVER: given a set $\mathcal{G}$, n subsets of $\mathcal{G}$, $S_1, S_2, \ldots, S_n$, and an integer k, do there exist k of those subsets whose union is $\mathcal{G}$?

SAT: SAT stands for "Boolean satisfiability." Given n propositional variables $p_1, p_2, \ldots, p_n$ and a formula in them with only the operators $\wedge, \vee, \neg$ and parenthesis ( ), does there exist an assignment of true or false to each of $p_1, \ldots, p_n$ that causes the formula to evaluate to true?

For example, the formula

$$((p_1 \wedge \neg p_2) \vee \neg p_1) \wedge p_3 \wedge \neg(p_4 \wedge p_2)$$

is satisfiable. A satisfying assigment is

$$p_1 = p_4 = 0, p_2 = p_3 = 1$$

The following formula is not satisfiable.

$$((p_1 \wedge \neg p_2) \vee (p_2 \wedge \neg p_1)) \wedge ((\neg p_1 \wedge \neg p_2) \vee (p_1 \wedge p_2))$$

Aside: I got the above formula by manipulating:

$$(p_1 \vee p_2) \wedge (\neg p_1 \vee p_2) \wedge (p_1 \vee \neg p_2) \wedge (\neg p_1 \vee \neg p_2)$$

Some special cases of SAT of particular interest to us:

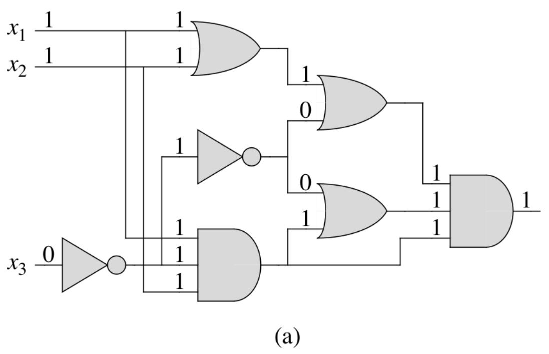
- CNF-SAT: the formula is a conjunction of *clauses*. Each clause is a disjunction of *literals*. A literal is either a variable or its negation. Example:

$$(p_1 \vee p_2 \vee \neg p_3) \wedge (\neg p_1) \wedge (p_1 \vee \neg p_2 \vee p_4 \vee p_5 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2)$$

- Exactly-3-CNF-SAT: a formula in CNF in which each clause has exactly 3 literals.
- At-Most-3-CNF-SAT: a formula in CNF in which each clause has at most 3 literals.
- DNF-SAT: the formula is a disjunction of clauses, each of which is a conjunction of literals. Example:

$$(p_1 \wedge p_2 \wedge \neg p_3) \vee (\neg p_1) \vee (p_1 \wedge \neg p_2 \wedge p_4 \wedge p_5 \wedge \neg p_3) \vee (\neg p_1 \wedge \neg p_2)$$

<u>CIRCUIT-SAT</u>: given a *combinational acyclic boolean circuit* of only AND, OR and NOT gates, n inputs wires and one output wire, does there exist an assignment of 0 or 1 to each of the input wires that causes the output wire to be 1? Following are two examples from your textbook (CLRS).



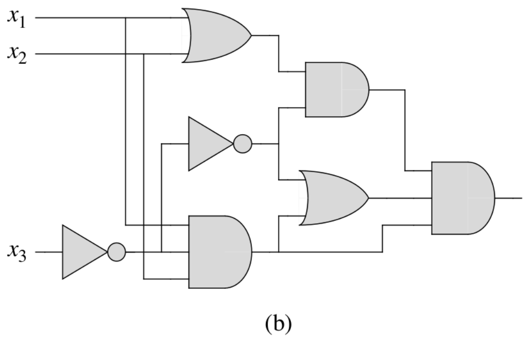


Figure 34.8 Two instances of the circuit-satisfiability problem. **(a)** The assignment $\langle x_1 = 1, x_2 = 1, x_3 = 0 \rangle$ to the inputs of this circuit causes the output of the circuit to be 1. The circuit is therefore satisfiable. **(b)** No assignment to the inputs of this circuit can cause the output of the circuit to be 1. The circuit is therefore unsatisfiable.

Example of decision problems that appear to not possess either of the two features:

(1) Given as input two digital circuits as in Circuit-SAT above, C_1, C_2, are they equivalent? That is, do they return the same output on every inputs?

Note that the *complement* of the above problem possesses both features. Given two digital circuits, are they not equivalent? If indeed for some C_1, C_2 this complementary question is true, then an evidence is an input i for which $C_1(i) \neq C_2(i)$.

(2) Given a situation on a chess board and it is Black's turn, is there a move that Black can make that guarantees that it will win?

(3) Given an undirected graph G, two distinct vertices a, b in it and an integer k, do there exist k distinct simple paths $a \rightsquigarrow b$ in G?

Note that algorithms exist for each of the problems (1)–(3) above. The only question is whether they appear to demonstrate the two features of interest to us.