Suppose you have an algorithm that can decide:

Given a TM M, does M halt on input 001?

Tell me how you can use this algorithm to solve this problem:

Given M, w: Does M halt on input w?

Assignment #5: Available from class web page. Due Friday Dec. 2 at the beginning of class.

Final exam tutorial:
Sunday Dec. 4 at 12:00pm in ECS 116.

There will be tutorials next week (the last week of classes).

Class P

A decision problem (yes/no question) is *in the class P* if there is a polynomial time algorithm for solving it.

Polynomial time: $O(n^c)$ for some constant c.

If a problem is solvable in polynomial time for

- some sensible encoding of the input
- some reasonable machine (TM/RAM/PC)

it can be solved in polynomial time for all other sensible encodings/reasonable machines.

A problem Q in NP is NP-complete if the existence of a polynomial time algorithm for Q implies the existence of a polynomial time algorithm for all problems in NP.

SAT (Satisfiability)

Variables: $u_1, u_2, u_3, \ldots u_k$.

A **literal** is a variable u_i or the negation of a variable $\neg u_i$.

If u is set to *true* then ¬ u is *false* and if u is set to *false* then ¬ u is *true*.

A **clause** is a set of literals. A clause is *true* if at least one of the literals in the clause is *true*.

The input to SAT is a collection of clauses.

This SAT problem has solution
u_1=T, u_2=F, u_3= T, u_4=F

(u_1 OR u_2 OR u_4) AND (¬ u_2 OR u_4) AND
(¬ u_1 OR u_3) AND (¬ u_4 OR ¬ u_1)

Does this SAT problem have a solution?

(u_1 OR u_2) AND (¬ u_2 OR u_3) AND

(¬ u_3 OR ¬ u_1) AND (¬ u_2 OR ¬ u_3) AND

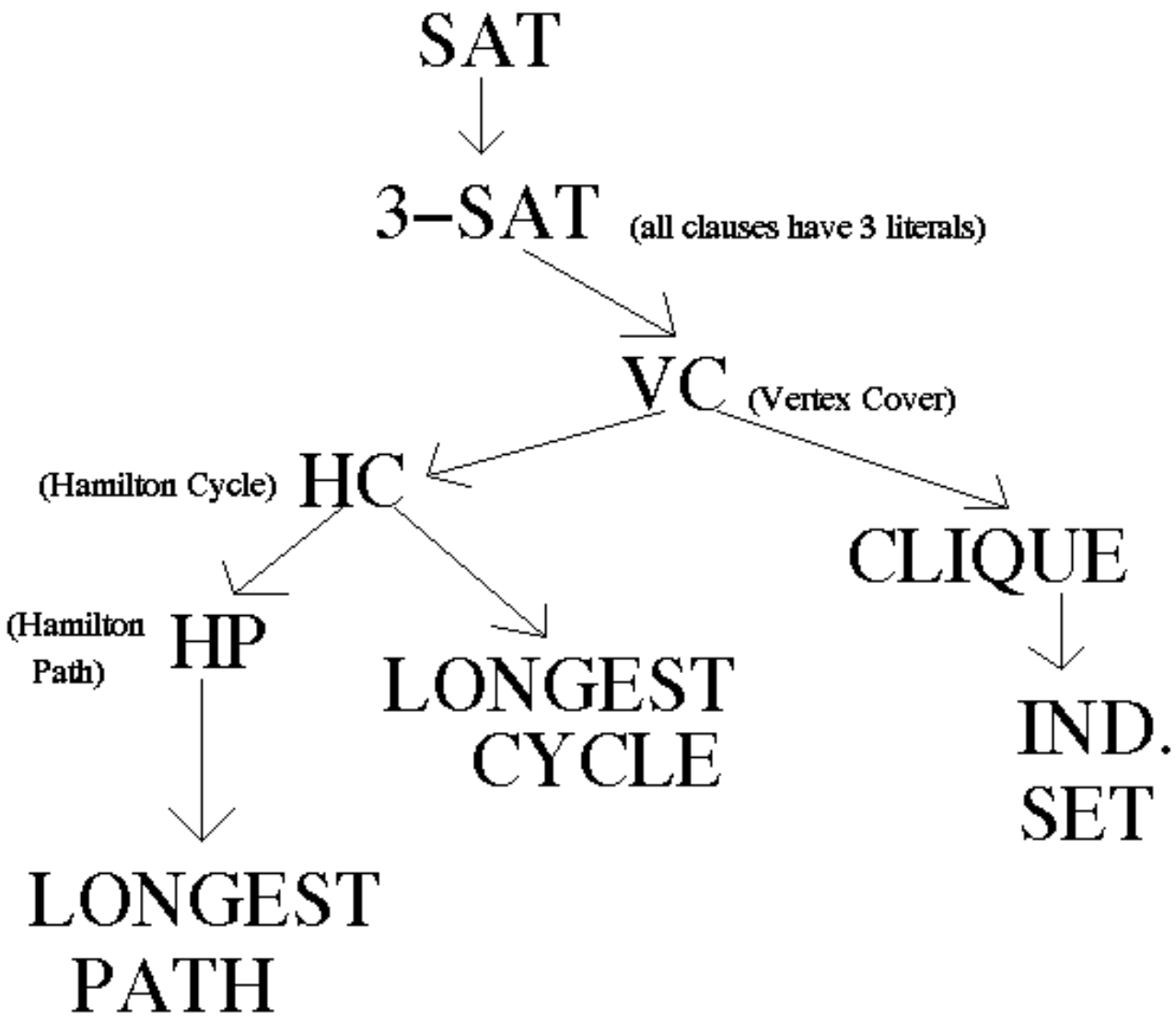
(u_3 OR ¬ u_1)

SAT (Satisfiability)

The output is the answer to: Is there an assignment of *true/false* to the variables so that every clause is satisfied (**satisfied** means the clause is *true*)?

If the answer is yes, such an assignment of the variables is called a **truth assignment**.

SAT is in NP: Certificate is true/false value for each variable in satisfying assignment.



History of NP−completeness Reductions

3-SAT- each clause must contain exactly 3 variables (assignment- at most 3).

Given: SAT is NP-complete (proof later)

Theorem: **3-SAT is NP-Complete.**

The first step in any NP-completeness proof is to argue that the problem is in NP.

The problem 3-SAT is a yes/no question.

Certificate: truth assignment, can be checked in polynomial time.

Next, we show that a polynomial time algorithm for 3-SAT implies the existence of one for SAT.

To convert a SAT problem to 3-SAT:

1. Clauses of size 1.

SAT: $\{z\}$

3-SAT:

$$\{z, y_1, y_2\},$$
$$\{z, \neg y_1, y_2\},$$
$$\{z, y_1, \neg y_2\},$$
$$\{z, \neg y_1, \neg y_2\}$$

y_1 and y_2 are new variables.

2. Clauses of size 2.

SAT: $\{z_1, z_2\}$

3-SAT: $\{z_1, z_2, y\}$,
$\{z_1, z_2, \neg y\}$

y is a new variable.

3. Clauses of size 3.

Leave these as they are since they are already acceptable for 3-SAT.

4.Clauses of size 4 or more.

SAT: $\{z_1, z_2, z_3, \ldots z_k\}$, $k>3$

3-SAT:

$\{ z_1, z_2, y_1\}$,
$\{\neg y_1, z_3, y_2\}$,
$\{\neg y_2, z_4, y_3\}$,
...
$\{\neg y_{k-4}, z_{k-2}, y_{k-3}\}$,
$\{\neg y_{k-3}, z_{k-1}, z_k\}$

$y_1, y_2, \ldots y_{k-3}$, are new variables.

This does not constitute a proof of NP-completeness unless we can argue that the size of the new 3-SAT problem problem is polynomially bounded by the size of the old SAT problem. Consider each case:

Size of clause	# new literals	size before	size after
1	2	1	12
2	1	2	6
3	0	3	3
$k \geq 4$	k-3	k	k + 2(k-3)

In all cases, the size after is at most 12 times the original problem size.

2-SAT: All clauses have at most 2 literals.

There is a linear time algorithm for 2-SAT

so 2-SAT is in P.

The 3-SAT problem is as hard as SAT but unless P=NP, 2-SAT is easier than 3-SAT or SAT.