## Circuits

A circuit is a directed graph $G = (V, E)$, with $V = \{1, \ldots, n\}$ together with a labeling: $l : V \to \{\mathtt{true}, \mathtt{false}, \wedge, \vee, \neg\}$, satisfying:

- If there is an edge $(i, j)$, then $i < j$;
- Every node in $V$ has *indegree* at most 2.
- A node $v$ has
  indegree 0 iff $l(v) \in \{\mathtt{true}, \mathtt{false}\}$;
  indegree 1 iff $l(v) = \neg$;
  indegree 2 iff $l(v) \in \{\vee, \wedge\}$

The value of the expression is given by the value at node $n$.





## CVP

A circuit is a more compact way of representing a Boolean expression.

Identical subexpressions need not be repeated.

CVP - the *circuit value problem* is, given a circuit, determine the value of the result node $n$.

CVP is solvable in polynomial time, by the algorithm which examines the nodes in increasing order, assigning a value `true` or `false` to each node.





## Composites

Consider the decision problem (or *language*) Composite defined by:

$$\{x \mid x \text{ is not prime}\}$$

This is the complement of the language Prime.

Is Composite $\in$ P?

Clearly, the answer is yes if, and only if, Prime $\in$ P.





## Hamiltonian Graphs

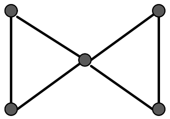
Given a graph $G = (V, E)$, a *Hamiltonian cycle* in $G$ is a path in the graph, starting and ending at the same node, such that every node in $V$ appears on the cycle *exactly once*.

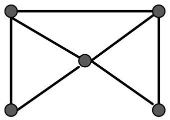
A graph is called *Hamiltonian* if it contains a Hamiltonian cycle.

The language HAM is the set of encodings of Hamiltonian graphs.

Is HAM $\in$ P?

## Examples





The first of these graphs is not Hamiltonian, but the second one is.





## Polynomial Verification

The problems Composite, SAT and HAM have something in common.

In each case, there is a *search space* of possible solutions.

> the factors of $x$; a truth assignment to the variables of $\phi$; a list of the vertices of $G$.

The number of possible solutions is *exponential* in the length of the input.

Given a potential solution, it is *easy* to check whether or not it is a solution.





## Verifiers

A verifier $V$ for a language $L$ is an algorithm such that

$$L = \{x \mid (x, c) \text{ is accepted by } V \text{ for some } c\}$$

If $V$ runs in time polynomial in the length of $x$, then we say that

> $L$ is *polynomially verifiable*.

Many natural examples arise, whenever we have to construct a solution to some design constraints or specifications.





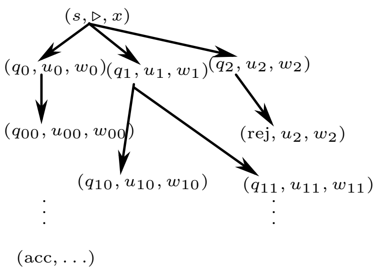
## Nondeterministic Complexity Classes

We have already defined $\mathsf{TIME}(f(n))$ and $\mathsf{SPACE}(f(n))$.

$\mathsf{NTIME}(f(n))$ is defined as the class of those languages $L$ which are accepted by a *nondeterministic* Turing machine $M$, such that for every $x \in L$, there is an accepting computation of $M$ on $x$ of length at most $f(n)$.

$$\mathsf{NP} = \bigcup_{k=1}^{\infty} \mathsf{NTIME}(n^k)$$

# Nondeterminism




For a language in $\mathsf{NTIME}(f(n))$, the height of the tree is bounded by $f(n)$ when the input is of length $n$.





# NP

*A language $L$ is polynomially verifiable if, and only if, it is in* $\mathsf{NP}$.

To prove this, suppose $L$ is a language, which has a verifier $V$, which runs in time $p(n)$.

The following describes a *nondeterministic algorithm* that accepts $L$

1. input $x$ of length $n$
2. nondeterministically guess $c$ of length $\leq p(n)$
3. run $V$ on $(x, c)$





# NP

In the other direction, suppose $M$ is a nondeterministic machine that accepts a language $L$ in time $n^k$.

We define the *deterministic algorithm* $V$ which on input $(x, c)$ simulates $M$ on input $x$.

At the $i^{\text{th}}$ nondeterministic choice point, $V$ looks at the $i^{\text{th}}$ character in $c$ to decide which branch to follow.

If $M$ accepts then $V$ accepts, otherwise it rejects.

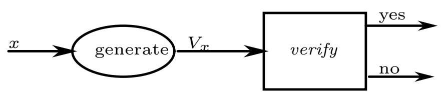
$V$ is a polynomial verifier for $L$.





# Generate and Test

We can think of nondeterministic algorithms in the generate-and test paradigm:




Where the *generate* component is nondeterministic and the *verify* component is deterministic.