# Graphs
# Cook – Levin Theorem

Stepan Kuznetsov

Discrete Math Bridging Course, HSE University

# The P Class

- Now we discuss only **decision problems:** that is, algorithmic questions with a "yes/no" answer.

# The P Class

- Now we discuss only **decision problems:** that is, algorithmic questions with a "yes/no" answer.
- For convenience, let the input data be a word over an alphabet: $x \in \Sigma^*$.

# The P Class

- Now we discuss only **decision problems:** that is, algorithmic questions with a “yes/no” answer.
- For convenience, let the input data be a word over an alphabet: $x \in \Sigma^*$.
- The size of input, $|x|$ is the length of $x$ in symbols.

# The P Class

- Now we discuss only **decision problems:** that is, algorithmic questions with a “yes/no” answer.
- For convenience, let the input data be a word over an alphabet: $x \in \Sigma^*$.
- The size of input, $|x|$ is the length of $x$ in symbols.
- A decision problem is in the P class, if there exists an algorithm for solving it, whose **worst case** running time is bounded by $p(|x|)$.

# The NP Class

- There are several equivalent definitions of the NP class.

# The NP Class

- There are several equivalent definitions of the NP class.
- Def. 1: non-deterministic computations.

# The NP Class

- There are several equivalent definitions of the NP class.
- Def. 1: non-deterministic computations.
    - The computation process may **branch:** at some point of execution, there could be more than one (but a finite number of) possibilities to perform the next step.

# The NP Class

- There are several equivalent definitions of the NP class.
- Def. 1: non-deterministic computations.
    - The computation process may **branch:** at some point of execution, there could be more than one (but a finite number of) possibilities to perform the next step.
    - **Angelic choice:** if at least one execution trajectory yields "yes," then the answer is "yes."

# The NP Class

- There are several equivalent definitions of the NP class.
- Def. 1: non-deterministic computations.
    - The computation process may **branch:** at some point of execution, there could be more than one (but a finite number of) possibilities to perform the next step.
    - **Angelic choice:** if at least one execution trajectory yields “yes,” then the answer is “yes.”
    - One can implement **non-deterministic guess** (say, guess the satisfying assignment for a 3-CNF or guess a Hamiltonian cycle in a graph).

# NP-Completeness

- **m-reduction** (Carp reduction): $A$ is reducible to $B$ ($A \leq_m^P B$), if there exists a polytime computable function $f\colon \Sigma^* \to \Sigma^*$, such that $\boxed{A(x) = 1 \iff B(f(x)) = 1.}$

# NP-Completeness

- **m-reduction** (Carp reduction): $A$ is reducible to $B$ ($A \leq_m^P B$), if there exists a polytime computable function $f\colon \Sigma^* \to \Sigma^*$, such that $\boxed{A(x) = 1 \iff B(f(x)) = 1.}$
- The idea of reduction: if we can solve $B$, we can also solve $A$: $A(x) = B(f(x))$.

# NP-Completeness

- **m-reduction** (Carp reduction): $A$ is reducible to $B$ ($A \leq_m^P B$), if there exists a polytime computable function $f\colon \Sigma^* \to \Sigma^*$, such that $\boxed{A(x) = 1 \iff B(f(x)) = 1.}$
- The idea of reduction: if we can solve $B$, we can also solve $A$: $A(x) = B(f(x))$.
- A problem $B$ is **NP-hard** if $A \leq_m^P B$ for any $A \in \text{NP}$.

# NP-Completeness

- **m-reduction** (Carp reduction): $A$ is reducible to $B$ ($A \leq_m^P B$), if there exists a polytime computable function $f\colon \Sigma^* \to \Sigma^*$, such that $\boxed{A(x) = 1 \iff B(f(x)) = 1.}$
- The idea of reduction: if we can solve $B$, we can also solve $A$: $A(x) = B(f(x))$.
- A problem $B$ is **NP-hard** if $A \leq_m^P B$ for any $A \in$ NP.
- $B$ is **NP-complete** if $B \in$ NP and $B$ is NP-hard.

# Cook – Levin Theorem

**Theorem**

*SAT (satisfiability of arbitrary Boolean formulae) NP-complete, that is, if $A \in$ NP, then $A$ is $m$-reducible to SAT.*

# Example: Graph Coloring

- Let us consider an example of an NP problem and show how it can be reduced to SAT.

# Example: Graph Coloring

- Let us consider an example of an NP problem and show how it can be reduced to SAT.
- The problem is **3-colorability of graphs.**

# Example: Graph Coloring

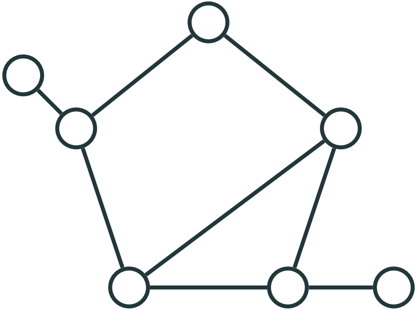
- Let us consider an example of an NP problem and show how it can be reduced to SAT.
- The problem is **3-colorability of graphs.**
- Let us first recall what a graph is.

# Graphs

An **undirected graph** is a formed by set of *vertices*, some of which are connected by *edges*.

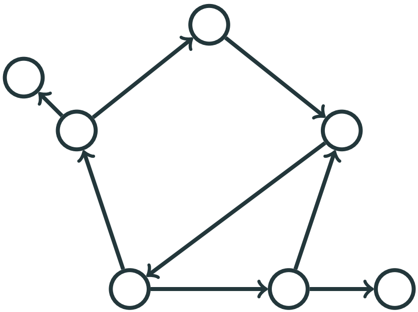
# Graphs

An **undirected graph** is a formed by set of *vertices*, some of which are connected by *edges*.



# Graphs

In a **directed** graph, edges have arrows on them:



# Loops and Parallel Edges

- Beware of two special kinds of edges in graphs.

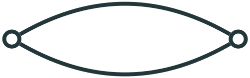
# Loops and Parallel Edges

- Beware of two special kinds of edges in graphs.
- *Loop:* a vertex connected to itself.

# Loops and Parallel Edges

- Beware of two special kinds of edges in graphs.
- *Loop:* a vertex connected to itself.

- *Parallel edges:* two vertices connected by more than one edge.



# Loops and Parallel Edges

- By default, loops and parallel edges are disallowed.

# Loops and Parallel Edges

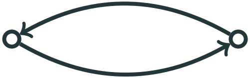
- By default, loops and parallel edges are **disallowed.**
- A graph with parallel edges is called a **multigraph.**

# Loops and Parallel Edges

- By default, loops and parallel edges are disallowed.
- A graph with parallel edges is called a **multigraph.**
- A graph with parallel edges and loops is called a **pseudograph.**

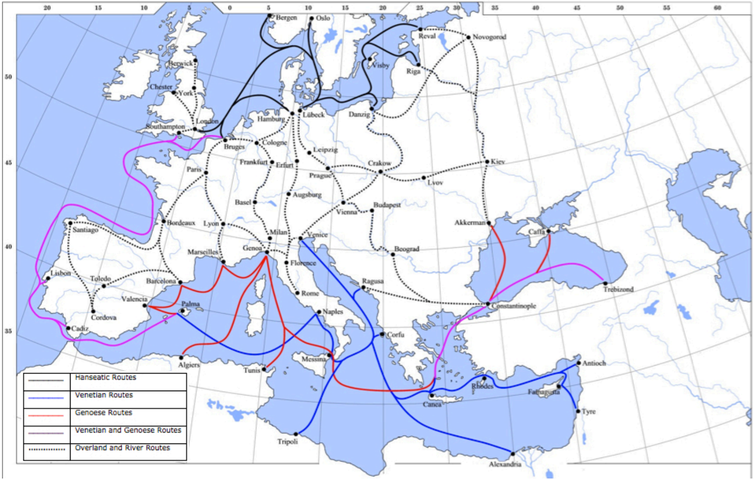
# Loops and Parallel Edges

- By default, loops and parallel edges are disallowed.
- A graph with parallel edges is called a **multigraph.**
- A graph with parallel edges and loops is called a **pseudograph.**
- Note that in a directed graph edges connecting two vertices in different directions are **not** considered parallel.



# Applications of Graphs

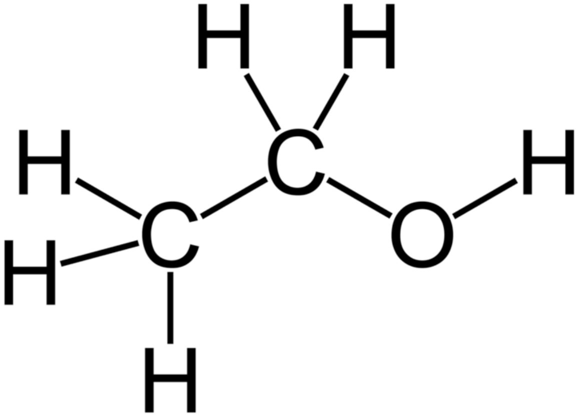
Maps (GIS): vertices = cities, edges = routes.




Lampman @ Wikipedia

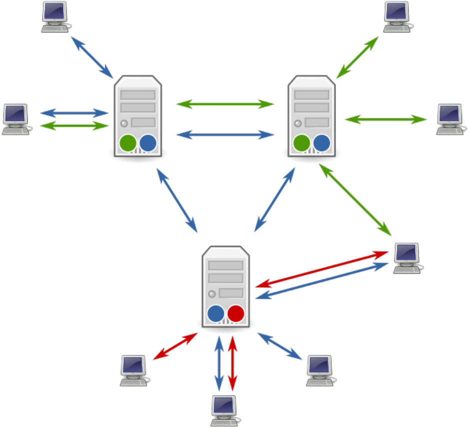
# Applications of Graphs

Chemistry: graphs of molecular structure.



# Applications of Graphs

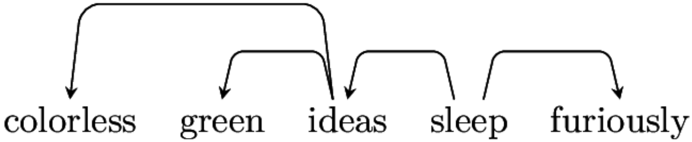
Internet: network topology.



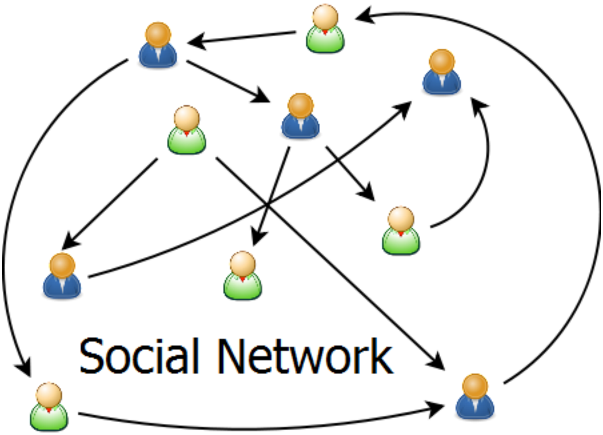
Benjamin D. Esham @ Wikipedia

# Applications of Graphs

Linguistics: syntactic dependencies.



# Applications of Graphs




Zigomitros Athanasios – Thor4bp @ Wikipedia

# Graph: Formal Definition

- A pseudograph can be formally defined as $G = (V, E)$, where $V$ is the set of vertices (arbitrary finite set) and $E \subseteq V \times V = \{(u, v) \mid u, v \in V\}$ is the set of edges, such that $(u, v) \in E \iff (v, u) \in E$.

# Graph: Formal Definition

- A pseudograph can be formally defined as $G = (V, E)$, where $V$ is the set of vertices (arbitrary finite set) and $E \subseteq V \times V = \{(u, v) \mid u, v \in V\}$ is the set of edges, such that $(u, v) \in E \iff (v, u) \in E$.
  - In other words, a pseudograph is a symmetric binary relation on a finite set $V$.

# Graph: Formal Definition

- A pseudograph can be formally defined as $G = (V, E)$, where $V$ is the set of vertices (arbitrary finite set) and $E \subseteq V \times V = \{(u, v) \mid u, v \in V\}$ is the set of edges, such that $(u, v) \in E \iff (v, u) \in E$.
  - In other words, a pseudograph is a symmetric binary relation on a finite set $V$.
  - An undirected graph is a symmetric irreflexive relation: $(u, u) \notin E$ for any $u$.

# Graph: Formal Definition

- A pseudograph can be formally defined as $G = (V, E)$, where $V$ is the set of vertices (arbitrary finite set) and $E \subseteq V \times V = \{(u, v) \mid u, v \in V\}$ is the set of edges, such that $(u, v) \in E \iff (v, u) \in E$.
  - In other words, a pseudograph is a symmetric binary relation on a finite set $V$.
  - An undirected graph is a symmetric irreflexive relation: $(u, u) \notin E$ for any $u$.
  - A directed graph is an arbitrary irreflexive relation.

# Graph: Formal Definition

- A pseudograph can be formally defined as $G = (V, E)$, where $V$ is the set of vertices (arbitrary finite set) and $E \subseteq V \times V = \{(u, v) \mid u, v \in V\}$ is the set of edges, such that $(u, v) \in E \iff (v, u) \in E$.
    - In other words, a pseudograph is a symmetric binary relation on a finite set $V$.
    - An undirected graph is a symmetric irreflexive relation: $(u, u) \notin E$ for any $u$.
    - A directed graph is an arbitrary irreflexive relation.
    - The formal definition of multigraph is more involved.

# Coloring

- We color vertices into colors of a set $C$, and our coloring is correct, if each edge connects vertices of different colors.

# Coloring

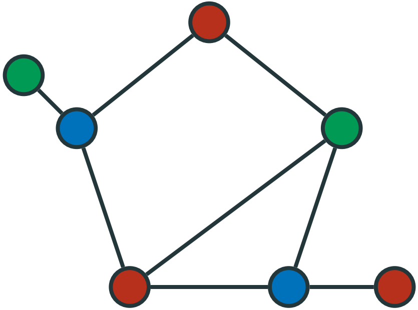
- We color vertices into colors of a set $C$, and our coloring is correct, if each edge connects vertices of different colors.
- Formally, $c\colon V \to C$, and if $(u, v) \in E$, then $c(u) \neq c(v)$.

# Coloring

- We color vertices into colors of a set $C$, and our coloring is correct, if each edge connects vertices of different colors.
- Formally, $c\colon V \to C$, and if $(u, v) \in E$, then $c(u) \neq c(v)$.
- Example: 3-coloring $c\colon V \to \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$.

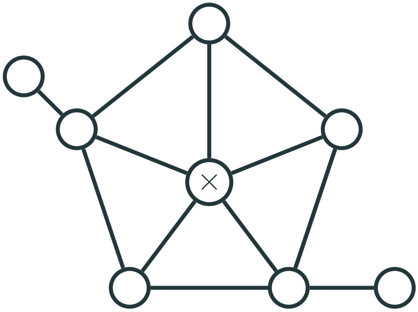
# 3-Coloring

For example, this graph is 3-colorable:



# 3-Coloring

… and this one is not:



# 3-Coloring $\in$ NP

- The 3-colorability problem (given a graph $G$, answer whether it is 3-colorable) clearly belongs to the NP class.

# 3-Coloring $\in$ NP

- The 3-colorability problem (given a graph $G$, answer whether it is 3-colorable) clearly belongs to the NP class.
- Indeed, we can non-deterministically guess the coloring ("hint") and then check its correctness in poly time.

# 3-Coloring $\in$ NP

- The 3-colorability problem (given a graph $G$, answer whether it is 3-colorable) clearly belongs to the NP class.
- Indeed, we can non-deterministically guess the coloring (“hint”) and then check its correctness in poly time.
- Thus, a particular case of Cook – Levin theorem states that 3-COLOR $\leq_m^P$ SAT.

# Reducing 3-COLOR to SAT

- For any graph $G$ we can construct a Boolean formula $\varphi_G$, which is satisfiable if and only if $G$ is 3-colorable.

# Reducing 3-COLOR to SAT

- For any graph $G$ we can construct a Boolean formula $\varphi_G$, which is satisfiable if and only if $G$ is 3-colorable.
- The reducing function $f\colon G \mapsto \varphi_G$ will be poly-time computable.

# Reducing 3-COLOR to SAT

- For any graph $G$ we can construct a Boolean formula $\varphi_G$, which is satisfiable if and only if $G$ is 3-colorable.
- The reducing function $f: G \mapsto \varphi_G$ will be poly-time computable.
- Moreover, each correct 3-coloring of $G$ will correspond to a satisfying assignment of $\varphi_G$, and vice versa.

# Reducing 3-COLOR to SAT

- For each vertex $v_i \in V$, introduce the following Boolean variables:
  $r_i$ "$v_i$ is colored red"
  $g_i$ "$v_i$ is colored green"
  $b_i$ "$v_i$ is colored blue"

# Reducing 3-COLOR to SAT

- For each vertex $v_i \in V$, introduce the following Boolean variables:
  - $r_i$ "$v_i$ is colored red"
  - $g_i$ "$v_i$ is colored green"
  - $b_i$ "$v_i$ is colored blue"
- $\varphi_G$ will represent natural conditions on these propositions.

# Reducing 3-COLOR to SAT

$$\varphi_G = \bigwedge_{v_i \in V} \big((r_i \vee g_i \vee b_i) \wedge$$

$$(\neg r_i \vee \neg g_i) \wedge (\neg r_i \vee \neg b_i) \wedge (\neg b_i \vee \neg g_i)\big) \wedge$$

$$\bigwedge_{(v_i, v_k) \in E} \big((\neg r_i \vee \neg r_k) \wedge (\neg g_i \vee \neg g_k) \wedge (\neg b_i \vee \neg b_k)\big)$$

# Reducing 3-COLOR to SAT

$$\varphi_G = \bigwedge_{v_i \in V} ((r_i \lor g_i \lor b_i) \land$$
$$(\neg r_i \lor \neg g_i) \land (\neg r_i \lor \neg b_i) \land (\neg b_i \lor \neg g_i)) \land$$
$$\bigwedge_{(v_i, v_k) \in E} ((\neg r_i \lor \neg r_k) \land (\neg g_i \lor \neg g_k) \land (\neg b_i \lor \neg b_k))$$

- 3-colorings of $G$ and satisfying assignments of $\varphi_G$ are in one-to-one correspondence.

# Reducing 3-COLOR to SAT

$$\varphi_G = \bigwedge_{v_i \in V} ((r_i \lor g_i \lor b_i) \land$$

$$(\neg r_i \lor \neg g_i) \land (\neg r_i \lor \neg b_i) \land (\neg b_i \lor \neg g_i)) \land$$

$$\bigwedge_{(v_i, v_k) \in E} ((\neg r_i \lor \neg r_k) \land (\neg g_i \lor \neg g_k) \land (\neg b_i \lor \neg b_k))$$

- 3-colorings of $G$ and satisfying assignments of $\varphi_G$ are in one-to-one correspondence.
- By the way, $\varphi_G$ is a 3-CNF.

# Reducing 3-COLOR to SAT

- Thus, if SAT were solvable in poly time, so would have been 3-COLOR.

# Reducing 3-COLOR to SAT

- Thus, if SAT were solvable in poly time, so would have been 3-COLOR.
- In reality, however, we **do not know** a polynomial algorithm for SAT, and such reductions give some evidence **against** its existence.

# Reducing 3-COLOR to SAT

- Thus, if SAT were solvable in poly time, so would have been 3-COLOR.
- In reality, however, we **do not know** a polynomial algorithm for SAT, and such reductions give some evidence **against** its existence.
- The idea of Cook – Levin theorem is that **any** NP guessing can be represented as guessing a satisfying assignment for a Boolean formula.

# Reducing 3-COLOR to SAT

- Reductions to SAT also yield positive results.

# Reducing 3-COLOR to SAT

- Reductions to SAT also yield positive results.
- While there is no known polynomial algorithm for SAT, modern SAT solvers are quite efficient in practice.

# Reducing 3-COLOR to SAT

- Reductions to SAT also yield positive results.
- While there is no known polynomial algorithm for SAT, modern **SAT solvers** are quite efficient in practice.
  - One of the reasons is that we measure **worst case** complexity, and instances which appear in practice could avoid such cases.

# Reducing 3-COLOR to SAT

- Reductions to SAT also yield positive results.
- While there is no known polynomial algorithm for SAT, modern **SAT solvers** are quite efficient in practice.
    - One of the reasons is that we measure **worst case** complexity, and instances which appear in practice could avoid such cases.
- Cook – Levin style reductions allow to use SAT solvers for other NP problems.

# Cook – Levin Theorem

**Theorem**

*SAT is NP-complete, that is, if $A \in$ NP, then $A$ is $m$-reducible to SAT.*

# Turing Machines

- In order to prove Cook – Levin theorem, we need to show that $A \leq_m^P$ SAT for **any** $A \in$ NP.

# Turing Machines

- In order to prove Cook – Levin theorem, we need to show that $A \leq_m^P$ SAT for **any** $A \in$ NP.
- This requires a formal notion of what an algorithm is, that is, a formal **model of computation.**

# Turing Machines

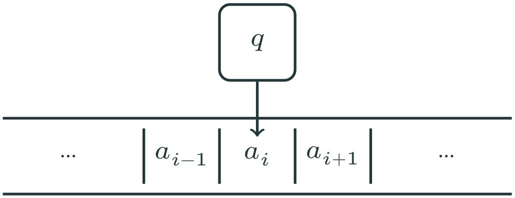
- In order to prove Cook – Levin theorem, we need to show that $A \leq_m^P$ SAT for **any** $A \in$ NP.
- This requires a formal notion of what an algorithm is, that is, a formal **model of computation.**
- Let us define one such model, namely, **Turing machines.**

# Turing Machines

- A Turing machine is a tuple $\mathfrak{M} = \langle \Sigma, \Gamma, Q, q_0, q_F, \Delta \rangle$, where:
  - $\Sigma$ is the *external alphabet* (in which input and output are formulated);
  - $\Gamma \supseteq \Sigma$ is the *internal alphabet* (used in the computation process);
  - $Q$ is a finite set of *states*;
  - $q_0$ is the starting state and $q_F$ is the final one;
  - $\Delta$ is the set of *rules* (also finite).

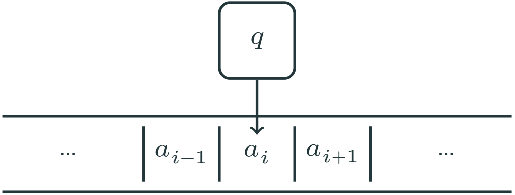
# Turing Machines

- At each step of its run, the machine keeps one of the states (from $Q$) in its internal memory, and observes one of the cells of an infinite tape:



# Turing Machines

- At each step of its run, the machine keeps one of the states (from $Q$) in its internal memory, and observes one of the cells of an infinite tape:




  - At each moment, only a finite part of the tape is populated by meaningful symbols; the rest is padded by "blank" symbols $B \in \Gamma - \Sigma$.

# Turing Machines

- **Rules** of $\mathfrak{M}$ (elements of $\Delta$) are of the form $\langle p, a \rangle \to \langle q, b, d \rangle$, where $p, q \in Q$, $a, b \in \Gamma$, and $d \in \{L, R, N\}$.

# Turing Machines

- **Rules** of $\mathfrak{M}$ (elements of $\Delta$) are of the form $\langle p, a\rangle \to \langle q, b, d\rangle$, where $p, q \in Q$, $a, b \in \Gamma$, and $d \in \{L, R, N\}$.
- Such a rule is executed as follows. If $\mathfrak{M}$ keeps $p$ in its internal memory and observes $a$ on the tape, then the following move is performed:
    1. replace $a$ with $b$ in the cell;
    2. replace $p$ with $q$ in the internal memory;
    3. if $d = L$, move one cell left; if $d = R$, move one cell right; if $d = N$, stay on the same cell.

# Turing Machines

- $\mathfrak{M}$ is **deterministic**, if for any $p \in Q$ and $a \in \Gamma$ there is at most one rule of the form $\langle p, a \rangle \to \ldots$

# Turing Machines

- $\mathfrak{M}$ is **deterministic,** if for any $p \in Q$ and $a \in \Gamma$ there is at most one rule of the form $\langle p, a \rangle \rightarrow \ldots$
- A deterministic machine, on a given input, has a unique execution trajectory; in general, the trajectory may branch.

# Turing Machines

- $\mathfrak{M}$ is **deterministic**, if for any $p \in Q$ and $a \in \Gamma$ there is at most one rule of the form $\langle p, a \rangle \to \ldots$
- A deterministic machine, on a given input, has a unique execution trajectory; in general, the trajectory may branch.
- Once a machine runs into state $q_F$, it stops successfully, and the word on the tape is the output.

# Turing Machines

- $\mathfrak{M}$ is **deterministic**, if for any $p \in Q$ and $a \in \Gamma$ there is at most one rule of the form $\langle p, a \rangle \to \ldots$
- A deterministic machine, on a given input, has a unique execution trajectory; in general, the trajectory may branch.
- Once a machine runs into state $q_F$, it stops successfully, and the word on the tape is the output.
- It is also possible to stop unsuccessfully or to run infinitely long.

# NP and co-NP

- A non-deterministic Turing machine solving a decision problem “accepts” the input, if it stops successfully and yields “yes” at **at least one** trajectory.

# NP and co-NP

- A non-deterministic Turing machine solving a decision problem “accepts” the input, if it stops successfully and yields “yes” at **at least one** trajectory.
- Thus, the *complement* of an NP-problem (say, non-satisfiability) is, in general, not in NP itself.

# NP and co-NP

- A non-deterministic Turing machine solving a decision problem “accepts” the input, if it stops successfully and yields “yes” at **at least one** trajectory.
- Thus, the *complement* of an NP-problem (say, non-satisfiability) is, in general, not in NP itself.
- This class is called co-NP.

# NP and co-NP

- A non-deterministic Turing machine solving a decision problem “accepts” the input, if it stops successfully and yields “yes” at **at least one** trajectory.
- Thus, the *complement* of an NP-problem (say, non-satisfiability) is, in general, not in NP itself.
- This class is called co-NP.
- Example: SAT is in NP, while TAUT (checking whether a Boolean formula is a tautology) is in co-NP.

# Church – Turing Thesis

- Turing machines form a complete computational model, by the following **Church – Turing thesis:** *any computation on a reasonable computing device can be performed on a Turing machine.*

# Church – Turing Thesis

- Turing machines form a complete computational model, by the following Church – Turing thesis: *any computation on a reasonable computing device can be performed on a Turing machine.*
- Moreover, if the computation is polynomial, it can be performed also polynomially on the Turing machine.

# Church – Turing Thesis

- Turing machines form a complete computational model, by the following **Church – Turing thesis:** *any computation on a reasonable computing device can be performed on a Turing machine.*
- Moreover, if the computation is polynomial, it can be performed also polynomially on the Turing machine.
    - The degree of the polynomial could change.

# Cook – Levin Theorem

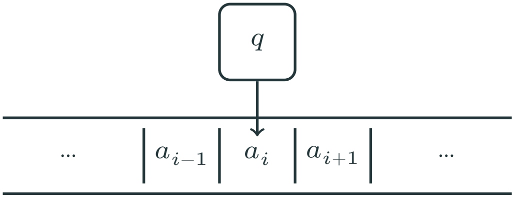
**Theorem**

*SAT is NP-complete, that is, if $A \in$ NP, then $A$ is $m$-reducible to SAT.*

# Cook – Levin Theorem

Proof sketch.

- Suppose $A \in$ NP, let us show $A \leq_m^P$ SAT.
- We encode each configuration of the non-deterministic Turing machine for $A$ as a binary word:




$$0^m \quad a_1 \quad \ldots \quad 0^m \quad a_{i-1} \quad q \quad a_i \quad 0^m \quad a_{i+1} \quad \ldots$$

# Cook – Levin Theorem

- The sequence of configurations (protocol) of $A$ on input $x$ is encoded by a binary matrix $(b_{ij})$ of size $(m \cdot p(|x|)) \times p(|x|)$.

# Cook – Levin Theorem

- The sequence of configurations (protocol) of $A$ on input $x$ is encoded by a binary matrix $(b_{ij})$ of size $(m \cdot p(|x|)) \times p(|x|)$.
- Next, we construct a formula $\varphi_x$ with variables $b_{00}, b_{01}, \ldots$ which expresses the fact that this matrix represents a correct protocol of a successful execution.

# Cook – Levin Theorem

$\varphi_x$ is a conjunction of the following claims:

1. the first row represents the configuration with $x$ on the tape, the machine observing its first letter;
2. each next row is obtained from the previous one by one of the rules of the machine;
3. the last row includes state $q_F$ and the answer “yes” (1).

# Cook – Levin Theorem

$\varphi_x$ is a conjunction of the following claims:

1. the first row represents the configuration with $x$ on the tape, the machine observing its first letter;
2. each next row is obtained from the previous one by one of the rules of the machine;
3. the last row includes state $q_F$ and the answer “yes” (1).

This is all expressible as Boolean formulae.

# Cook – Levin Theorem

- The reducing function is $f\colon x \mapsto \varphi_x$.

# Cook – Levin Theorem

- The reducing function is $f\colon x \mapsto \varphi_x$.
- $A(x) = 1 \iff \varphi_x$ is satisfiable.

# Cook – Levin Theorem

- The reducing function is $f\colon x \mapsto \varphi_x$.
- $A(x) = 1 \iff \varphi_x$ is satisfiable.
- Thus, $A \leq_m^P$ SAT.

# Cook – Levin Theorem

- The reducing function is $f\colon x \mapsto \varphi_x$.
- $A(x) = 1 \iff \varphi_x$ is satisfiable.
- Thus, $A \leq_m^P$ SAT.
- Since $A$ was taken arbitrarily, we get NP-hardness of SAT.

# Cook – Levin Theorem

- The reducing function is $f\colon x \mapsto \varphi_x$.
- $A(x) = 1 \iff \varphi_x$ is satisfiable.
- Thus, $A \leq_m^P$ SAT.
- Since $A$ was taken arbitrarily, we get NP-hardness of SAT.
- On the other hand, SAT is in NP, so it is NP-complete.

# NP-completeness of 3-SAT

- 3-SAT is a special version of SAT, where only 3-CNFs are allowed.

# NP-completeness of 3-SAT

- 3-SAT is a special version of SAT, where only 3-CNFs are allowed.
- Trivially, 3-SAT $\leq_m^P$ SAT... but we need the opposite reduction!

# NP-completeness of 3-SAT

- 3-SAT is a special version of SAT, where only 3-CNFs are allowed.
- Trivially, 3-SAT $\leq_m^P$ SAT... but we need the opposite reduction!
- Let us show that SAT $\leq_m^P$ 3-SAT.

# Tseitin's Transformations

**Theorem**

*For any Boolean formula $A$, there exists an equisatisfiable 3-CNF $B$ of polynomial size.*

- Equisatisfiability means that $B$ is satisfiable iff so is $A$.

# Tseitin's Transformations

**Theorem**

*For any Boolean formula $A$, there exists an equisatisfiable 3-CNF $B$ of polynomial size.*

- Equisatisfiability means that $B$ is satisfiable iff so is $A$.
- Constructing an *equivalent* 3-CNF of polynomial size is not always possible: even translation to CNF can lead to exponential blowup.

# Tseitin's Transformations

- Tseitin's transformations look like translation into 3-address (Assembler-like) code:
  $(a + b) * (c + d)$ is translated to
  "add $a$ $b$ $t_1$; add $c$ $d$ $t_2$; mul $t_1$ $t_2$ $r$"

# Tseitin's Transformations

- Tseitin's transformations look like translation into 3-address (Assembler-like) code:
  $(a + b) * (c + d)$ is translated to
  "add $a$ $b$ $t_1$; add $c$ $d$ $t_2$; mul $t_1$ $t_2$ $r$"
- For each subformula we introduce a new variable and write the corresponding equivalences.

# Tseitin's Transformations

Example: $(p \to q) \vee (q \to (p \to r))$

# Tseitin's Transformations

Example: $(p \to q) \lor (q \to (p \to r))$

$$
\begin{aligned}
&(t_1 \leftrightarrow (p \to q)) \land \\
&(t_2 \leftrightarrow (p \to r)) \land \\
&(t_3 \leftrightarrow (q \to t_2)) \land \\
&(t_4 \leftrightarrow (t_1 \lor t_3)) \land \\
&t_4
\end{aligned}
$$

# Tseitin's Transformations

Transform into 3-CNF by the following table:

| | |
|---|---|
| $t_k \leftrightarrow (t_i \wedge t_j)$ | $(\neg t_i \vee \neg t_j \vee t_k) \wedge (t_i \vee \neg t_k) \wedge (t_j \vee \neg t_k)$ |
| $t_k \leftrightarrow (t_i \vee t_j)$ | $(t_i \vee t_j \vee \neg t_k) \wedge (\neg t_i \vee t_k) \wedge (\neg t_j \vee t_k)$ |
| $t_k \leftrightarrow (t_i \rightarrow t_j)$ | $(\neg t_i \vee t_j \vee \neg t_k) \wedge (t_i \vee t_k) \wedge (\neg t_j \vee t_k)$ |
| $t_k \leftrightarrow \neg t_i$ | $(t_i \vee t_k) \wedge (\neg t_i \vee \neg t_k)$ |

# Tseitin's Transformations

Transform into 3-CNF by the following table:

| | |
|---|---|
| $t_k \leftrightarrow (t_i \wedge t_j)$ | $(\neg t_i \vee \neg t_j \vee t_k) \wedge (t_i \vee \neg t_k) \wedge (t_j \vee \neg t_k)$ |
| $t_k \leftrightarrow (t_i \vee t_j)$ | $(t_i \vee t_j \vee \neg t_k) \wedge (\neg t_i \vee t_k) \wedge (\neg t_j \vee t_k)$ |
| $t_k \leftrightarrow (t_i \rightarrow t_j)$ | $(\neg t_i \vee t_j \vee \neg t_k) \wedge (t_i \vee t_k) \wedge (\neg t_j \vee t_k)$ |
| $t_k \leftrightarrow \neg t_i$ | $(t_i \vee t_k) \wedge (\neg t_i \vee \neg t_k)$ |

For our example, we get:

$$(\neg p \vee q \vee \neg t_1) \wedge (p \vee t_1) \wedge (\neg q \vee t_1) \wedge$$
$$(\neg p \vee r \vee \neg t_2) \wedge (p \vee t_2) \wedge (\neg r \vee t_2) \wedge$$
$$(\neg q \vee t_2 \vee \neg t_3) \wedge (q \vee t_3) \wedge (\neg t_2 \vee t_3) \wedge$$
$$(t_1 \vee t_3 \vee \neg t_4) \wedge (\neg t_1 \vee t_4) \wedge (\neg t_3 \vee t_4) \wedge t_4$$

# Beyond Decision Problems

- An NP decision problem is the question **whether there exists** a *witness* $y$ such that $R(x, y) = 1$.

# Beyond Decision Problems

- An NP decision problem is the question **whether there exists** a *witness* $y$ such that $R(x, y) = 1$.
    - E.g., a satisfying assignment for $\varphi$.

# Beyond Decision Problems

- An NP decision problem is the question **whether there exists** a *witness* $y$ such that $R(x, y) = 1$.
  - E.g., a satisfying assignment for $\varphi$.
- **Search problem:** yield a witness or say “no.”

# Beyond Decision Problems

- An NP decision problem is the question **whether there exists** a *witness* $y$ such that $R(x, y) = 1$.
  - E.g., a satisfying assignment for $\varphi$.
- **Search problem:** yield a witness or say "no."
- **Counting problem** (the #P class): yield the number of witnesses.

# Beyond Decision Problems

- An NP decision problem is the question **whether there exists** a *witness* $y$ such that $R(x, y) = 1$.
  - E.g., a satisfying assignment for $\varphi$.
- **Search problem:** yield a witness or say "no."
- **Counting problem** (the #P class): yield the number of witnesses.
- Finally, we could ask for **all** witnesses.

# Beyond Decision Problems

- The problem of yielding all witnesses could be unconditionally non-polynomial, since the answer could be exponential.

# Beyond Decision Problems

- The problem of yielding all witnesses could be unconditionally non-polynomial, since the answer could be exponential.
- If $P = NP$, then any search problem is solvable in poly time.

# Beyond Decision Problems

- The problem of yielding all witnesses could be unconditionally non-polynomial, since the answer could be exponential.
- If P = NP, then any search problem is solvable in poly time.
  - Dichotomy: take $\varphi[p_1 := 0]$ and $\varphi[p_1 := 1]$, find out which is satisfiable, then do the same for $p_2, p_3, ...$.

# Beyond Decision Problems

- The problem of yielding all witnesses could be unconditionally non-polynomial, since the answer could be exponential.
- If P $=$ NP, then any search problem is solvable in poly time.
  - Dichotomy: take $\varphi[p_1 := 0]$ and $\varphi[p_1 := 1]$, find out which is satisfiable, then do the same for $p_2, p_3, ...$.
  - This gives a poly-time algorithm for the search problem for 2-CNF.

# Beyond Decision Problems

- The problem of yielding all witnesses could be unconditionally non-polynomial, since the answer could be exponential.
- If P $=$ NP, then any search problem is solvable in poly time.
  - Dichotomy: take $\varphi[p_1 := 0]$ and $\varphi[p_1 := 1]$, find out which is satisfiable, then do the same for $p_2, p_3, \ldots$.
  - This gives a poly-time algorithm for the search problem for 2-CNF.
- The counting problem could be harder than the decision one (example: DNF-SAT).