Constraint Satisfaction Problems
Mathematical Background: Sets, Relations, and Graphs

Bernhard Nebel, Julien Hué, and Stefan Wölfl

Albert-Ludwigs-Universität Freiburg

April 23, 25, and 2012; May 2, 2012
Formal definition of CSP uses sets and constraints

Constraints are specific relations that restrict possible solutions

CSP solving techniques use operations that manipulate sets and relations

CSP instances can also be represented by various kinds of graphs

Graph-theoretical notions can be used to describe, e.g., structural properties of constraint networks

Complexity for solving CSP instances can depend on both the relations used in the constraints and properties of the constraint graphs
Set-theoretical notions
Sets:

Naive understanding: a set is a “well-defined” collection of objects.

Principles/Set-theoretical axioms (ZF):
Axioms that describe which objects count as sets and which operations can be used to form new sets ...
Sets:

Naive understanding: a set is a “well-defined” collection of objects.

Principles/Set-theoretical axioms (ZF):
Axioms that describe which objects count as sets and which operations can be used to form new sets...
Set theory

Some set-theoretical axioms (ZF):

- **Extensionality**: Two sets are equal if and only if they contain the same elements.
- **Empty set**: There is a set, ∅, with no elements.
- **Pairs**: For any pair of sets \( x, y \), \( \{ x, y \} \) is a set.
- **Union**: For any set \( x \), there exists a set, \( \bigcup x \), whose elements are precisely the elements of the elements of \( x \).
- **Separation**: For any set \( x \) and any property \( F(y) \), there is a subset of \( x \), \( \{ y \in x : F(y) \} \), containing precisely the elements \( y \) of \( x \) for which \( F(y) \) holds.
- **Power set**: For any set \( x \) there exists a set \( 2^x \) such that the elements of \( 2^x \) are precisely the subsets of \( x \).
- **Axiom of choice**: Given a set \( x \) of pairwise disjoint nonempty sets, there is a set \( y \) that contains exactly one element from each member of \( x \).
Set-theoretical notations

Usually, we argue naïvely by using the following notations . . .

Boolean operations on sets:

\[ A \cup B := \{ x : x \in A \text{ or } x \in B \} \]
\[ A \cap B := \{ x \in A : x \in B \} \]
\[ A \setminus B := \{ x \in A : x \notin B \} \]

Subset relation: \( A \subseteq B, A \subsetneq B, \text{etc.}, \) are defined as usual.

Power set: \( 2^A := \{ B : B \subseteq A \} \)

(Ordered) pairs:

\[ (x, y) := \{ \{ x \}, \{ x, y \} \} \]
\[ (x_1, \ldots, x_n) := ((x_1, \ldots, x_{n-1}), x_n) \]

Product: \( A \times B := \{ (a, b) : a \in A \text{ and } b \in B \} \)
A **Boolean algebra (with complements)** is a set \( A \) with

- two binary operations \( \sqcap, \sqcup \),
- a unary operation \( - \), and
- two distinct elements 0 and 1

such that for all elements \( a, b \) and \( c \) of \( A \):

\[
\begin{align*}
    a \sqcup (b \sqcup c) &= (a \sqcup b) \sqcup c & a \sqcap (b \sqcap c) &= (a \sqcap b) \sqcap c & \text{Ass} \\
    a \sqcup b &= b \sqcup a & a \sqcap b &= b \sqcap a & \text{Com} \\
    a \sqcup (a \sqcap b) &= a & a \sqcap (a \sqcup b) &= a & \text{Abs} \\
    a \sqcup (b \sqcap c) &= (a \sqcup b) \sqcap (a \sqcup c) & a \sqcap (b \sqcup c) &= (a \sqcap b) \sqcup (a \sqcap c) & \text{Dis} \\
    a \sqcup -a &= 1 & a \sqcap -a &= 0 & \text{Compl}
\end{align*}
\]
Set algebras

Definition

A set algebra on a set $X$ is a non-empty subset $\mathcal{F}$ of $2^X$ that is closed under unions, intersections, and complements. $\langle X, \mathcal{F} \rangle$ is called a field of sets.

Notice: a set algebra on $X$ contains $X$ and $\emptyset$ as elements.

Lemma

(a) The power set of any set forms a set algebra.
(b) Each set algebra defines a Boolean algebra.
(c) A finite Boolean algebra can always be represented as a power set, . . .
(d) more generally, each Boolean algebra is isomorphic to a field of sets (Stone’s representation theorem).
Proof of the lemma.

(a) By applying complement, union, or intersection on subsets of a given set $X$, we again obtain subsets of $X$.

(b) A set algebra $\mathcal{F}$ on $X$ contains $\emptyset$ and $X$. $\overline{A} := X \setminus A$ is a unary operation on $\mathcal{F}$; $\cap$ and $\cup$ are binary operations. Hence, $\langle \mathcal{F}, \cap, \cup, \neg, \emptyset, X \rangle$ is a structure that obviously satisfies all properties of a Boolean algebra.

(c) One has to show: given a finite Boolean algebra $B = \langle A, \sqcap, \sqcup, \neg, 0, 1 \rangle$ there exists a set $X$ such that . . .

(next slide . . .)
Proof of the lemma (cont’d):

... $B$ and $2^X$ are **isomorphic** (as Boolean algebras).

1. Define a partial order on $B$:
   
   $a \leq b : \iff b \cap a = a$ ( $\iff b \cup a = b$ $\iff a \cap -b = 0$)
   
   $a < b : \iff a \leq b \land a \neq b$.

   The set of **atoms** (i.e., non-zero minimal element of $B$) is def. by:
   
   $At_B := \{ a \in A : 0 \leq a \land \text{there is no } b \in A \text{ s.t. } 0 < b < a \}$.

2. Define a map $f : A \rightarrow 2^{At_B}, x \mapsto \{ a \in At_B : a \leq x \}$.

   Obviously $f(a) = \{ a \}$ for each $a \in At_B$.

3. $f$ is an **homomorphism** of Boolean algebras, i.e., it preserves
   
   Boolean operations: $f(0) = \emptyset$, $f(1) = X$, $f(-x) = \overline{f(x)}$, $f(x \cap y) = f(x) \cap f(y)$, and $f(x \cup y) = f(x) \cup f(y)$.

4. $f$ is a **bijection**, i.e., it is injective (“one-to-one”) and surjective (“onto”).
Proof of the lemma (cont’d):

... $B$ and $2^X$ are isomorphic (as Boolean algebras).

1. Define a partial order on $B$:
   
   $a \leq b : \iff b \land a = a$ ( $\iff b \lor a = b$ $\iff a \land \neg b = 0$)
   
   $a < b : \iff a \leq b \land a \neq b$.

   The set of atoms (i.e., non-zero minimal element of $B$) is def. by:
   
   $At_B := \{a \in A : 0 \leq a \land \text{there is no } b \in A \text{ s.t. } 0 < b < a\}$.

2. Define a map $f : A \to 2^{At_B}$, $x \mapsto \{a \in At_B : a \leq x\}$.

   Obviously $f(a) = \{a\}$ for each $a \in At_B$.

3. $f$ is an homomorphism of Boolean algebras, i.e., it preserves Boolean operations:
   
   $f(0) = \emptyset$, $f(1) = X$, $f(-x) = \overline{f(x)}$,
   
   $f(x \land y) = f(x) \land f(y)$, and $f(x \lor y) = f(x) \lor f(y)$.

4. $f$ is a bijection, i.e., it is injective (“one-to-one”) and surjective (“onto”).
Proof of the lemma (cont’d):

... $B$ and $2^X$ are isomorphic (as Boolean algebras).

1. Define a partial order on $B$:
   
   $a \leq b : \iff b \cap a = a$ ( $\iff b \cup a = b$ $\iff a \cap \neg b = 0$)
   
   $a < b : \iff a \leq b$ and $a \neq b$.

   The set of atoms (i.e., non-zero minimal element of $B$) is def. by:
   $\text{At}_B := \{a \in A : 0 \leq a \land \text{there is no } b \in A \text{ s.t. } 0 < b < a\}$.

2. Define a map $f : A \rightarrow 2^{\text{At}_B}$, $x \mapsto \{a \in \text{At}_B : a \leq x\}$.

   Obviously $f(a) = \{a\}$ for each $a \in \text{At}_B$.

3. $f$ is an homomorphism of Boolean algebras, i.e., it preserves Boolean operations: $f(0) = \emptyset$, $f(1) = X$, $f(\neg x) = \overline{f(x)}$, $f(x \cap y) = f(x) \cap f(y)$, and $f(x \cup y) = f(x) \cup f(y)$.

4. $f$ is a bijection, i.e., it is injective ("one-to-one") and surjective ("onto").
Proof of the lemma (cont’d):

... $B$ and $2^X$ are isomorphic (as Boolean algebras).

1. Define a partial order on $B$:
   
   \[ a \leq b : \iff b \cap a = a \quad (\iff b \cup a = b \iff a \cap \neg b = 0) \]
   
   \[ a < b : \iff a \leq b \land a \neq b. \]

   The set of atoms (i.e., non-zero minimal element of $B$) is def. by:
   
   \[ \text{At}_B := \{a \in A : 0 \leq a \land \text{there is no } b \in A \text{ s.t. } 0 < b < a\}. \]

2. Define a map $f : A \rightarrow 2^{\text{At}_B}$, $x \mapsto \{a \in \text{At}_B : a \leq x\}$.
   
   Obviously $f(a) = \{a\}$ for each $a \in \text{At}_B$.

3. $f$ is an homomorphism of Boolean algebras, i.e., it preserves Boolean operations: $f(0) = \emptyset$, $f(1) = X$, $f(\neg x) = \overline{f(x)}$, $f(x \cap y) = f(x) \cap f(y)$, and $f(x \cup y) = f(x) \cup f(y)$.

4. $f$ is a bijection, i.e., it is injective ("one-to-one") and surjective ("onto").
Relations
A relation over sets $X_1, \ldots, X_n$ is a subset

$$R \subseteq X_1 \times \cdots \times X_n =: \prod_{1 \leq i \leq n} X_i.$$ 

The number $n$ is referred to as arity of $R$. An $n$-ary relation on a set $X$ is a subset

$$R \subseteq X^n := X \times \cdots \times X \quad (n \text{ times}).$$

Since relations are sets, set-theoretical operations (union, intersection, complement) can be applied to relations as well.
Binary relations

For binary relations on a set \( X \) we have some special operations:

**Definition**

Let \( R, S \) be binary (2-ary) relations on \( X \). The **converse** of relation \( R \) is defined by:

\[
R^{-1} := \left\{ (x, y) \in X^2 : (y, x) \in R \right\}.
\]

The **composition** of relations \( R \) and \( S \) is defined by:

\[
R \circ S := \left\{ (x, z) \in X^2 : \exists y \in X \text{ s.t. } (x, y) \in R \text{ and } (y, z) \in S \right\}.
\]

The **identity relation** is:

\[
\Delta_X := \left\{ (x, y) \in X^2 : x = y \right\}.
\]
Lemma

Let $X$ be a non-empty set. Let $\mathcal{R}(X)$ be the set of all binary relations on $X$. Then:

(a) $\mathcal{R}(X)$ is a set algebra on $X \times X$.

(b) For all relations $R, S, T \in \mathcal{R}(X)$:

\[
R \circ (S \circ T) = (R \circ S) \circ T
\]

\[
R \circ (S \cup T) = (R \circ S) \cup (R \circ T)
\]

\[
\Delta_X \circ R = R \circ \Delta_X = R
\]

\[
(R^{-1})^{-1} = R \quad \text{and} \quad (-R)^{-1} = -(R^{-1})
\]

\[
(R \cup S)^{-1} = R^{-1} \cup S^{-1}
\]

\[
(R \circ S)^{-1} = S^{-1} \circ R^{-1}
\]

\[
(R \circ S) \cap T^{-1} = \emptyset \quad \text{if and only if} \quad (S \circ T) \cap R^{-1} = \emptyset
\]
Constraints, relations, and variables

Constraints can be expressed by relations that restrict value assignments to variables.

Consider variables $x_1, x_2, x_3$ and relations $B, C$ defined by:

$$B = \{(x, y, z) \in [0..3]^3 : x < y < z\}$$

$$C = \{(x, y, z) \in [0..3]^3 : x > y > z\}.$$

- "$(x_1, x_2, x_3)$ satisfies $B$" and "$(x_3, x_2, x_1)$ satisfies $B$" express different constraints, while . . .

- "$(x_3, x_2, x_1)$ satisfies $B$" and "$(x_1, x_2, x_3)$ satisfies $C$" essentially express the same constraint.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_3$</th>
<th>$x_2$</th>
<th>$x_1$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Let $V$ be a set of variables. For $v \in V$, let $\text{dom}(v)$ be a non-empty set (of values), called the domain of $v$.

**Definition**

A relation over (pairwise distinct) variables $v_1, \ldots, v_n \in V$ is a pair

$$R_{v_1, \ldots, v_n} := ((v_1, \ldots, v_n), R)$$

where $R$ is a relation over $\text{dom}(v_1), \ldots, \text{dom}(v_n)$.

The sequence $(v_1, \ldots, v_n)$ is referred to as the scheme (or: range), the set $\{v_1, \ldots, v_n\}$ as the scope, and $R$ as the graph of $R_{v_1, \ldots, v_n}$.

We will not always distinguish between a relation over variables and its graph (and between scope and scheme), e.g., we write

$$R_{v_1, \ldots, v_n} \subseteq \text{dom}(v_1) \times \cdots \times \text{dom}(v_n).$$
Let $R_{\bar{v}} = (\bar{v}, R)$ be a relation over variables $\bar{v} = (v_1, \ldots, v_n)$.

**Definition**

For any fixed values $a_1 \in \text{dom}(v_{i_1}), \ldots, a_k \in \text{dom}(v_{i_k})$, define

$$
\sigma_{v_{i_1}=a_1, \ldots, v_{i_k}=a_k}(\bar{v}, R) := (\bar{v}, R')
$$

with

$$
R' := \{ (x_1, \ldots, x_n) \in R : x_{i_j} = a_j, \text{ for each } 1 \leq j \leq k \}.
$$

The (unary) operation $\sigma_{v_{i_1}=a_1, \ldots, v_{i_k}=a_k}$ is called **selection or restriction**.
Let \((i_1, \ldots, i_k)\) be a \(k\)-tuple of pairwise distinct elements of \(\{1, \ldots, n\}\) \((k \leq n)\).

**Definition**

Given a relation \((\overline{v}, R)\) over \(\overline{v}\),

\[
\pi_{v_{i_1}, \ldots, v_{i_k}} (\overline{v}, R) := ((v_{i_1}, \ldots, v_{i_k}), R')
\]

with

\[
R' := \left\{ \overline{y} \in \prod_{1 \leq j \leq k} \text{dom}(v_{i_j}) : \overline{y} = (x_{i_1}, \ldots, x_{i_k}), \right. \\
\left. \text{for some } (x_1, \ldots, x_n) \in R \right\}
\]

is a relation over \((v_{i_1}, \ldots, v_{i_k})\), called the projection of \((\overline{v}, R)\) on \((v_{i_1}, \ldots, v_{i_k})\).

Note: Each permutation of the scheme \(\overline{v}\) defines a projection. For binary relations \(R = R_{x,y}, R^{-1} = \pi_{y,x}(R_{x,y})\).
... Joins

Definition

Consider pairwise distinct variables $v_1, \ldots, v_n$.

Let $(\overline{v}, R)$ and $(\overline{v}', S)$ be relations over variables

$\overline{v} = (v_{i_1}, \ldots, v_{i_k})$ and $\overline{v}' = (v_{j_1}, \ldots, v_{j_l})$, resp., such that

$\{v_{i_1}, \ldots, v_{i_k}\} \cup \{v_{j_1}, \ldots, v_{j_l}\} = \{v_1, \ldots, v_n\}$. Then

$$(\overline{v}, R) \bowtie (\overline{v}', S) := ((v_1, \ldots, v_n), T)$$

with

$$T = \left\{ \overline{x} \in \prod_{1 \leq i \leq n} \text{dom}(v_i) : (x_{i_1}, \ldots, x_{i_k}) \in R \text{ and } (x_{j_1}, \ldots, x_{j_l}) \in S \right\}$$

is a relation over $(v_1, \ldots, v_n)$, the join of $(\overline{v}, R)$ and $(\overline{v}', S)$.

For binary relations $R = R_{x,y}$ and $S = S_{y,z}$ on the same set,

$R \circ S = \pi_{x,z}(R_{x,y} \bowtie S_{y,z})$. 
Examples

Consider relations $R := R_{x_1,x_2,x_3}$ and $S := S_{x_2,x_3,x_4}$ defined by:

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td>$c$</td>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td>$c$</td>
<td>$n$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>$c$</td>
<td>2</td>
</tr>
<tr>
<td>$b$</td>
<td>$c$</td>
<td>3</td>
</tr>
</tbody>
</table>

Then $\sigma_{x_3=c}(R)$, $\pi_{x_2,x_3}(R)$, $\pi_{x_2,x_1}(R)$, and $R \bowtie S$ are:

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td>$c$</td>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td>$c$</td>
<td>$n$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td>$n$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_2$</th>
<th>$x_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td>$n$</td>
<td>$c$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>$c$</td>
<td>2</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
<td>$c$</td>
<td>3</td>
</tr>
<tr>
<td>$c$</td>
<td>$b$</td>
<td>$c$</td>
<td>2</td>
</tr>
<tr>
<td>$c$</td>
<td>$b$</td>
<td>$c$</td>
<td>3</td>
</tr>
</tbody>
</table>
Graphs
An (undirected, simple) graph is an ordered pair

\[ G := \langle V, E \rangle \]

where:
- \( V \) is a non-empty set (of vertices, nodes);
- \( E \) is a set of two-element subsets \( X \subseteq V \) (elements of \( E \) are called edges).

Usually, we assume that the graph (i.e., \(|V|\)) is finite.

In undirected, simple graphs edges are often written as \([u, v]\).

Sometimes, one allows \( E \) to also contain singleton subsets of \( V \) (loops), written as \([v, v]\). But simple graphs are always loopless.
An (undirected, simple) graph is an ordered pair

\[ G := (V, E) \]

where:

- \( V \) is a non-empty set (of vertices, nodes);
- \( E \) is a set of two-element subsets \( X \subseteq V \) (elements of \( E \) are called edges).

Usually, we assume that the graph (i.e., \(|V|\)) is finite.

In undirected, simple graphs edges are often written as \([u, v]\).

Sometimes, one allows \( E \) to also contain singleton subsets of \( V \) (loops), written as \([v, v]\). But simple graphs are always loopless.
An *(undirected, simple)* graph is an ordered pair

\[ G := \langle V, E \rangle \]

where:

- \( V \) is a non-empty set (of *vertices, nodes*);
- \( E \) is a set of two-element subsets \( X \subseteq V \) (elements of \( E \) are called *edges*).

Usually, we assume that the graph (i.e., \(|V|\)) is finite.

In undirected, simple graphs edges are often written as \([u, v]\).

Sometimes, one allows \( E \) to also contain singleton subsets of \( V \) (*loops*), written as \([v, v]\). But *simple* graphs are always loopless.
A simple undirected graph

![Undirected Graph Diagram]

- Nodes: E, A, B, C, D, F
- Edges: EA, AB, BC, CD, DE, EF, CF, FB, BD, AD, CE
Often we allow for multiple edges between the same set of end vertices.

**Definition**

An *(undirected, multi-)* graph is an ordered triple

\[ G := \langle V, E, \gamma \rangle \]

where:

- \( V \) is non-empty set (of *vertices, nodes*);
- \( \gamma : E \to \{ X \in 2^V : 1 \leq |X| \leq 2 \} \).

The elements of \( E \) are called *edges*.

We always assume: \( V \cap E = \emptyset \).

The *order* of a graph is the number of vertices \( |V| \). Often, \( |E| \) is referred to as the *size* of \( G \), but often we specify both \( n := |V| \) and \( m := |E| \).
Undirected multi-graph

Often we allow for multiple edges between the same set of end vertices.

Definition

An (undirected, multi-) graph is an ordered triple

\[ G := \langle V, E, \gamma \rangle \]

where:

- \( V \) is non-empty set (of vertices, nodes);
- \( \gamma : E \rightarrow \{ X \in 2^V : 1 \leq |X| \leq 2 \} \).

The elements of \( E \) are called edges.

We always assume: \( V \cap E = \emptyset \).

The order of a graph is the number of vertices \( |V| \). Often, \( |E| \) is referred to as the size of \( G \), but often we specify both \( n := |V| \) and \( m := |E| \).
An undirected multi-graph
Definition

Let $G = \langle V, E, \gamma \rangle$ be an undirected graph.

(a) If $\gamma(e) = \{u, v\}$ for some $e \in E$, then $u$ and $v$ are called adjacent (or: connected by $e$).

(b) A path (or: walk) in $G$ is a sequence

$$(v_0, e_1, v_1, \ldots, e_k, v_k)$$

such that $e_1, \ldots, e_k \in E$ and $\gamma(e_i) = \{v_{i-1}, v_i\}$ (for each $1 \leq i \leq k$). $k$ is referred to as length, $v_0$ as start vertex, and $v_k$ as end vertex of the path.

(c) A cycle is a path $(v_0, \ldots, e_k, v_k)$ with $v_0 = v_k$ and $k \geq 1$.

(d) A path $(v_0, \ldots, e_k, v_k)$ is simple if $e_i \neq e_j$ for all $i \neq j$.

(e) A path $(v_0, \ldots, e_k, v_k)$ is elementary if $v_1 \neq v_j$ for $0 \leq i \neq j \leq k$ (but $v_0 = v_k$ is allowed).
Paths: An example

Figure: A simple path visiting the nodes $B, A, E, D, F$
Let $G = \langle V, E, \gamma \rangle$ be an undirected graph.

Definition

(a) $G$ is **connected** if for each pair of vertices $u$ and $v$, there exists a path from $u$ to $v$.

(b) $G$ is **complete** if any pair of vertices is connected by an edge.

(c) $G$ is a **forest** if $G$ is cycle-free.

(d) $G$ is a **tree** if $G$ is cycle-free and connected.
Examples

Figure: Connected, but not complete
Examples

Figure: A not connected graph
Examples

Figure: A forest
Figure: A tree
Graph-theoretical notions

Let \( G = \langle V, E, \gamma \rangle \) be an undirected graph.

**Definition**

Let \( V' \) be a non-empty subset of \( V \). Then \( G[V'] = \langle V', E', \gamma' \rangle \) with:
\[
E' := \{ e \in E : \gamma(e) \subseteq V' \} \quad \text{and} \quad \gamma' := \gamma|_{E'}
\]
is called the **subgraph** induced by \( V' \).

**Definition**

Let \( E' \) be a subset of \( E \). Then \( G[E'] = \langle V', E', \gamma|_{E'} \rangle \) is called the **partial graph** induced by \( E' \).

**Definition**

A **clique** in a graph \( G \) is a complete subgraph of \( G \).
Let $G = \langle V, E, \gamma \rangle$ be an undirected graph.

**Definition**

Let $V'$ be a non-empty subset of $V$. Then $G[V'] = \langle V', E', \gamma' \rangle$ with:

$$E' := \{ e \in E : \gamma(e) \subseteq V' \} \quad \text{and} \quad \gamma' := \gamma|_{E'}$$

is called the subgraph induced by $V'$.

**Definition**

Let $E'$ be a subset of $E$. Then $G[E'] = \langle V', E', \gamma|_{E'} \rangle$ is called the partial graph induced by $E'$.

**Definition**

A clique in a graph $G$ is a complete subgraph of $G$. 
Let $G = \langle V, E, \gamma \rangle$ be an undirected graph.

**Definition**

Let $V'$ be a non-empty subset of $V$. Then $G[V'] = \langle V', E', \gamma' \rangle$ with:

$$E' := \{ e \in E : \gamma(e) \subseteq V' \} \text{ and } \gamma' := \gamma|_{E'}$$

is called the **subgraph** induced by $V'$.

**Definition**

Let $E'$ be a subset of $E$. Then $G[E'] = \langle V', E', \gamma|_{E'} \rangle$ is called the **partial graph** induced by $E'$.

**Definition**

A **clique** in a graph $G$ is a complete subgraph of $G$. 
Examples

Figure: A partial graph
Examples

Figure: A subgraph
Examples

Figure: A clique
A directed (multi-) graph (or: digraph) is an ordered tuple

\[ G := \langle V, A, \alpha, \omega \rangle \]

where:

- \( V \) is a non-empty set (of vertices or nodes),
- \( A \) is a set (elements of \( A \) are called arcs, edges, or arrows),
- \( \alpha, \omega : A \to V \) are functions.

\( \alpha(a) \) is called the start vertex of \( a \), \( \omega(a) \) the end vertex of \( a \).

If \( G \) has no parallel arcs (\( a, a' \in A \) with \( \alpha(a) = \alpha(a') \) and \( \omega(a) = \omega(a') \)), we can write \( A \) as a set of tuples:

\[ \{(\alpha(a),\omega(a)) \in V^2 : a \in A\} . \]

In that case we use the notation \( \langle V, A \rangle \) instead of \( \langle V, A, \alpha, \omega \rangle \).
Directed Graph

**Definition**

A directed (multi-) graph (or: digraph) is an ordered tuple

\[ G := \langle V, A, \alpha, \omega \rangle \]

where:

- \( V \) is a non-empty set (of vertices or nodes),
- \( A \) is a set (elements of \( A \) are called arcs, edges, or arrows),
- \( \alpha, \omega : A \rightarrow V \) are functions.

\( \alpha(a) \) is called the start vertex of \( a \), \( \omega(a) \) the end vertex of \( a \).

If \( G \) has no parallel arcs (\( a, a' \in A \) with \( \alpha(a) = \alpha(a') \) and \( \omega(a) = \omega(a') \)), we can write \( A \) as a set of tuples:

\[ \left\{ (\alpha(a), \omega(a)) \in V^2 : a \in A \right\} . \]

In that case we use the notation \( \langle V, A \rangle \) instead of \( \langle V, A, \alpha, \omega \rangle \).
Most notions introduced for undirected graphs can easily be adapted for directed graphs. For example:

**Definition**

A path in $G$ is a sequence $(v_0, a_1, v_1, \ldots, a_k, v_k)$ such that $a_1, \ldots, a_k \in A$ and for each $1 \leq i \leq k$, $\alpha(a_i) = v_{i-1}$ and $\omega(a_i) = v_i$.

$g^+(v)$: the outdegree of $v$, the number of arcs that start from $v$

$g^-(v)$: the indegree of $v$, the number of arcs that end in $v$

parents of $v$: nodes with an arc to $v$

childs of $v$: nodes with an arc from $v$
Digraphs: Some notions

Most notions introduced for undirected graphs can easily be adapted for directed graphs. For example:

**Definition**

A path in $G$ is a sequence $(v_0, a_1, v_1, \ldots, a_k, v_k)$ such that $a_1, \ldots, a_k \in A$ and for each $1 \leq i \leq k$, $\alpha(a_i) = v_{i-1}$ and $\omega(a_i) = v_i$.

$g^+(v)$: the outdegree of $v$, the number of arcs that start from $v$

$g^-(v)$: the indegree of $v$, the number of arcs that end in $v$

parents of $v$: nodes with an arc to $v$

childs of $v$: nodes with an arc from $v$
A directed multi-graph
A directed multi-graph

Figure: A directed graph with a (strongly) connected subgraph
Labeled graphs

Often graphs $G = \langle V, E/A, \ldots \rangle$ are equipped with labeling functions.

Let $L$ be a not-empty set of labels.

**Vertex labeling:** a function $l : V \rightarrow L$ that assigns to each $v$ a vertex label $l(v) \in L$.

**Edge labeling:** a function $l : E \rightarrow L$ that assigns to each $e \in E$ a label $l(v) \in L$.

Example: In route planning, one can represent street networks as digraphs with an arc labeling (expressing travelling distance between places/nodes).

The label set may be equipped with further structures. In the route planning example, the labeling function is understood as a distance function (metric space).
Often graphs $G = \langle V, E/A, \ldots \rangle$ are equipped with labeling functions.

Let $L$ be a not-empty set of labels. 

**Vertex labeling:** a function $l : V \rightarrow L$ that assigns to each $v$ a vertex label $l(v) \in L$.

**Edge labeling:** a function $l : E \rightarrow L$ that assigns to each $e \in E$ a label $l(v) \in L$.

Example: In route planning, one can represent street networks as digraphs with an arc labeling (expressing travelling distance between places/nodes).

The label set may be equipped with further structures. In the route planning example, the labeling function is understood as a distance function (metric space).
Often graphs $G = \langle V, E/A, \ldots \rangle$ are equipped with labeling functions.

Let $L$ be a not-empty set of labels.

**Vertex labeling**: a function $l : V \rightarrow L$ that assigns to each $v$ a vertex label $l(v) \in L$.

**Edge labeling**: a function $l : E \rightarrow L$ that assigns to each $e \in E$ a label $l(v) \in L$.

Example: In route planning, one can represent street networks as digraphs with an arc labeling (expressing travelling distance between places/nodes).

The label set may be equipped with further structures. In the route planning example, the labeling function is understood as a distance function (metric space).
Often graphs $G = \langle V, E/A, \ldots \rangle$ are equipped with labeling functions.

Let $L$ be a not-empty set of labels.

**Vertex labeling:** a function $l : V \rightarrow L$ that assigns to each $v$ a vertex label $l(v) \in L$.

**Edge labeling:** a function $l : E \rightarrow L$ that assigns to each $e \in E$ a label $l(v) \in L$.

Example: In route planning, one can represent street networks as digraphs with an arc labeling (expressing travelling distance between places/nodes).

The label set may be equipped with further structures. In the route planning example, the labeling function is understood as a distance function (metric space).
Graphs can be used to represent binary relations between nodes. For relations of higher arity we need:

**Definition**

A hypergraph is a pair $H := \langle V, E \rangle$, where

- $V$ is a set (of nodes, vertices),
- $E$ is a set of non-empty subsets of $V$ (called hyperedges), i.e., $E \subseteq 2^V \setminus \{\emptyset\}$.

Notice: Hyperedges may contain arbitrarily many nodes.

$k$-uniform hypergraph: each hyperedge contains exactly $k$ vertices.
Graphs can be used to represent binary relations between nodes. For relations of higher arity we need:

**Definition**

A hypergraph is a pair $H := \langle V, E \rangle$, where

- $V$ is a set (of nodes, vertices),
- $E$ is a set of non-empty subsets of $V$ (called hyperedges), i.e., $E \subseteq 2^V \setminus \{\emptyset\}$.

Notice: Hyperedges may contain arbitrarily many nodes.

$k$-uniform hypergraph: each hyperedge contains exactly $k$ vertices.
Hypergraphs: An example

Figure: A hypergraph
Computational Complexity
In the lecture we do not use a specific model of computation: any Turing-complete abstract machine (Turing machine, (unit cost) RAM, ...) suffices.

When analyzing algorithms, we use a **uniform cost model**: constant costs are assumed for every machine operation (regardless of the size of its input).

### Landau symbols

Let $M$ be the set of all functions $f : \mathbb{N} \rightarrow \mathbb{R}$, $g \in M$.

- $O(g) = \{ f \in M : \exists c \in \mathbb{R} \ \exists n_0 \in \mathbb{N} \ \forall n > n_0 : f(n) \leq c \cdot g(n) \}$
- $\Omega(g) = \{ f \in M : \exists c \in \mathbb{R} \ \exists n_0 \in \mathbb{N} \ \forall n > n_0 : f(n) \geq c \cdot g(n) \}$
- $\Theta(g) = O(g) \cap \Omega(g)$
In the lecture we do not use a specific model of computation: any Turing-complete abstract machine (Turing machine, (unit cost) RAM, . . . ) suffices.

When analyzing algorithms, we use a uniform cost model: constant costs are assumed for every machine operation (regardless of the size of its input).

**Landau symbols**

Let $M$ be the set of all functions $f : \mathbb{N} \rightarrow \mathbb{R}$, $g \in M$.

- $O(g) = \{ f \in M : \exists c \in \mathbb{R} \exists n_0 \in \mathbb{N} \forall n > n_0 : f(n) \leq c \cdot g(n) \}$
- $\Omega(g) = \{ f \in M : \exists c \in \mathbb{R} \exists n_0 \in \mathbb{N} \forall n > n_0 : f(n) \geq c \cdot g(n) \}$
- $\Theta(g) = O(g) \cap \Omega(g)$
Data structures

- Runtime depends on used data structures

- For example: basic operations on a graph depend on how the graph is represented (e.g., as an adjacency matrix or an adjacency list).

Let \( G = \langle V, A, \alpha, \omega \rangle \) be a digraph.

Adjacency matrix: \( n \times n \) matrix \( (a_{ij})_{1 \leq i, j \leq n} \) such that \( a_{ij} \) is the number of arcs from vertex \( v_i \) to vertex \( v_j \).

Adjacency list: an array of lists, namely, for each vertex \( v \), the list of \( v \)'s children (in undirected graphs: neighbors = adjacent vertices)
Data structures

- Runtime depends on used data structures
- For example: basic operations on a graph depend on how the graph is represented (e.g., as an adjacency matrix or an adjacency list).

Let $G = \langle V, A, \alpha, \omega \rangle$ be a digraph.

**Adjacency matrix**: $n \times n$ matrix $(a_{i,j})_{1 \leq i,j \leq n}$ such that $a_{i,j}$ is the number of arcs from vertex $v_i$ to vertex $v_j$.

**Adjacency list**: an array of lists, namely, for each vertex $v$, the list of $v$’s children (in undirected graphs: neighbors $=$ adjacent vertices)
Adjacency matrix

Graph:

Adjacency matrix:

\[
\begin{pmatrix}
0 & 1 & 0 & 0 & 2 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}
\]
Graph:

Adjacency list:

1 → (2, 5, 5)
2 → (3, 4, 6)
3 → ()
4 → (1, 2)
5 → (5)
6 → (4)
Comparing basic operations

Consider the following operations on a digraph (without parallel arcs):

- **Arc**: Check whether there is an arc from \( v \) to \( w \) \(((v, w) \in E?)\);
- **Deg\(^+\)**: Determine the outdegree of \( v \) \((g^+(v) = ?)\);
- **Root**: Check whether there exists a \( v \) with \( g^-(v) = 0 \).

<table>
<thead>
<tr>
<th>Data structure</th>
<th>Memory</th>
<th>Arc</th>
<th>Deg(^+)</th>
<th>Root</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjacency matrix</td>
<td>(\Theta(n^2))</td>
<td>(\mathcal{O}(1))</td>
<td>(\mathcal{O}(n))</td>
<td>(\mathcal{O}(n^2))</td>
</tr>
<tr>
<td>Adjacency list</td>
<td>(\Theta(n + m))</td>
<td>(\mathcal{O}(g^+(v)))</td>
<td>(\mathcal{O}(g^+(v)))</td>
<td>(\mathcal{O}(n + m))</td>
</tr>
</tbody>
</table>

\(n\): number of vertices; \(m\): number of arcs/edges
In the lecture we will study three types of computational problems:

- Decision problems
  - Expected output: YES/No
- Search problems
  - Expected output: a solution
- Optimization problems
  - Expected output: an optimal solution
Decision problem

Let $P$ be a set of problem instances and $F$ be a unary property defined on $P$.

Then the decision problem “$x$ satisfies $F$?” is defined as follows:

- **Given:** A problem instance $x \in P$
- **Question:** Does $x$ satisfy condition $F$?

**Example**

- **Given:** A digraph $G = \langle V, E \rangle$, vertices $v_1, v_2 \in V$.
- **Question:** Does there exist a path from $v_1$ to $v_2$ in $G$?
Decision problem

Let $P$ be a set of problem instances and $F$ be a unary property defined on $P$.
Then the decision problem “$x$ satisfies $F$?” is defined as follows:

- **Given**: A problem instance $x \in P$
- **Question**: Does $x$ satisfy condition $F$?

**Example**

- **Given**: A digraph $G = \langle V, E \rangle$, vertices $v_1, v_2 \in V$.
- **Question**: Does there exist a path from $v_1$ to $v_2$ in $G$?
Let $P$ be a set of problem instances, $S$ be the set of solutions, and $R$ be a binary relation $R \subseteq P \times S$.

Then the search problem “Find a solution of $x$?” is defined as follows:

- **Given:** A problem instance $x \in P$
- **Asked:** A solution $s \in S$ with $(x, s) \in R$

**Example**

- **Given:** A digraph $G = \langle V, E \rangle$, vertices $v_1, v_2 \in V$.
- **Asked:** Find a path from $v_1$ to $v_2$ in $G$ (if there exists one; otherwise “failure”)!
Search problem

Let $P$ be a set of problem instances, $S$ be the set of solutions, and $R$ be a binary relation $R \subseteq P \times S$.

Then the search problem “Find a solution of $x$?” is defined as follows:

- **Given**: A problem instance $x \in P$
- **Asked**: A solution $s \in S$ with $(x, s) \in R$

**Example**

- **Given**: A digraph $G = \langle V, E \rangle$, vertices $v_1, v_2 \in V$.
- **Asked**: Find a path from $v_1$ to $v_2$ in $G$ (if there exists one; otherwise “failure”)!
Optimization problem

Let $P$ be a set of problem instances, $S$ be the set of solutions, $R$ be a binary relation $R \subseteq P \times S$, and $f : S \rightarrow \mathbb{R}$ be an objective function.

The optimization problem “Find an optimal solution of $x$?” is defined as follows:

- **Given**: A problem instance $x \in P$
- **Asked**: A solution $s \in S$ with $(x, s) \in R$ that maximizes/minimizes $f$, i.e., $f(s)$ is maximal/minimal among all $s$ with $(x, s) \in R$.

**Example**

- **Given**: A weighted digraph $G = \langle V, E \rangle$, vertices $v_1, v_2 \in V$.
- **Asked**: Find a shortest path from $v_1$ to $v_2$ in $G$ (if there exists one; otherwise “failure”)!
Optimization problem

Let $P$ be a set of problem instances, $S$ be the set of solutions, $R$ be a binary relation $R \subseteq P \times S$, and $f : S \rightarrow \mathbb{R}$ be an objective function.

The optimization problem “Find an optimal solution of $x$?” is defined as follows:

- **Given:** A problem instance $x \in P$
- **Asked:** A solution $s \in S$ with $(x, s) \in R$ that maximizes/minimizes $f$, i.e., $f(s)$ is maximal/minimal among all $s$ with $(x, s) \in R$.

**Example**

- **Given:** A weighted digraph $G = \langle V, E \rangle$, vertices $v_1, v_2 \in V$.
- **Asked:** Find a shortest path from $v_1$ to $v_2$ in $G$ (if there exists one; otherwise “failure”)!
P: class of decision problems that can be solved by a deterministic Turing machine in polynomial time

NP: class of decision problems that can be solved by a non-deterministic Turing machine in polynomial time

Alternative characterization of NP:
NP: class of decision problems $P$ such that there exists a polynomial verifier for $P$.

A verifier for a decision problem $P$ is a procedure that, given a problem instance $x$ and a candidate solution $s$ (called certificate), verifies that $s$ is a solution of $x$.

A verifier is polynomial if it verifies $(x, s) \in R$ in polynomial time (it need not run in polynomial time on input $(x, s) \notin R$)
Consider decision problems $P$ and $P'$ encoded as formal languages $L, L'$ over alphabets $\Sigma, \Sigma'$.

**Polynomial reduction:** $L'$ is polynomially reducible to $L$, $L' \leq_p L$, if there exists a total and polynomial time-computable function $f : \Sigma' \rightarrow \Sigma$ such that $x \in L' \iff f(x) \in L$.

**Definition**

- A decision problem $L$ is **NP-hard** if for each decision problem $L'$ in NP, it holds $L' \leq_p L$.
- A decision problem $L$ is **NP-complete** if it is both in NP and NP-hard.
The Boolean satisfiability problem, i.e., the problem of deciding whether a propositional logic formula $\varphi$ is satisfiable, is NP-complete.

3CNF-SAT formula: a propositional logic formula $\varphi$ that is in conjunctive normal form such that each clause contains at most 3 literals.

The problem of deciding whether a 3CNF-SAT formula is satisfiable is NP-complete.
The problem **3-Colorability** is defined as follows: Given an undirected, simple graph $G = \langle V, E \rangle$, is there a vertex coloring $c : V \rightarrow \{1, 2, 3\}$ such that for each pair of adjacent vertices $v, v'$ in $G$, $c(v) \neq c(v')$.

**Theorem**

**3-Colorability** is NP-complete.

**Proof.**

Obviously, **3-Colorability** is in NP: we only need to guess the coloring $c$. Then we check whether this coloring assigns different colors to adjacent vertices. This can be done in polynomial time.

We now show that **3-Colorability** is NP-hard by a polynomial reduction from 3CNF-SAT. Since 3CNF-SAT is NP-complete, each problem in NP can be reduced to 3CNF-SAT and via 3CNF-SAT $\leq_p$ **3-Colorability**, each problem in NP can also be reduced to **3-Colorability**.
The problem **3-Colorability** is defined as follows: Given an undirected, simple graph $G = \langle V, E \rangle$, is there a vertex coloring $c : V \rightarrow \{1, 2, 3\}$ such that for each pair of adjacent vertices $v, v'$ in $G$, $c(v) \neq c(v')$.

**Theorem**

**3-Colorability** is NP-complete.

**Proof.**

Obviously, **3-Colorability** is in NP: we only need to guess the coloring $c$. Then we check whether this coloring assigns different colors to adjacent vertices. This can be done in polynomial time.

We now show that **3-Colorability** is NP-hard by a polynomial reduction from **3CNF-SAT**. Since **3CNF-SAT** is NP-complete, each problem in NP can be reduced to **3CNF-SAT** and via **3CNF-SAT** $\leq_p$ **3-Colorability**, each problem in NP can also be reduced to **3-Colorability**.
We construct a function that assigns to each 3CNF-SAT formula $\varphi = C_1 \land \cdots \land C_m$ a graph $G_\varphi$ such that

$\varphi$ is satisfiable $\iff$ $G_\varphi$ has a coloring with colors \{red, blue, green\}.

We assume (w.l.o.g.) that each clause $C_j$ consists of exactly three literals, i.e., $C_j = (l_{1j} \lor l_{j2} \lor l_{j3})$. Let $x_1, \ldots, x_n$ be the set of propositional variables that occur in $\varphi$. $G_\varphi$ will contain the following subgraph $G_T$ (with $2n + 1$ vertices):

![Diagram of 3-colorability graph]
For each clause $C_j$ ($1 \leq j \leq m$) we add a subgraph $G_j$ (clause gadget) with new vertices $a_j, b_j, c_j, y_j, z_j$ and a vertex $v$ which is the same in each of the clause gadgets:

![Diagram showing the addition of a subgraph $G_j$ with vertices $a_j, b_j, c_j, y_j, z_j$ and $v$.](image)

Vertices in $G_j$ are connected by an edge to vertices in $G_T$ as follows:

- an edge $\{u, v\}$
- edges $\{a_j, l_{j1}\}, \{b_j, l_{j2}\}, \{c_j, l_{j3}\}$ ($1 \leq j \leq m$)
For example, if $\varphi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots$, $G_\varphi$ contains the following subgraph $G_1$: 

![Diagram of graph $G_1$ showing nodes $u$, $x_1$, $x_2$, $x_4$, $\overline{x_1}$, $\overline{x_2}$, $\overline{x_4}$, $z_1$, $v$, $a_1$, $y_1$, $b_1$, $c_1$. Edges connect these nodes as specified by the logical proposition $\varphi$.]
For example, if $\varphi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots$, $G_\varphi$ contains the following subgraph $G_1$:

Assume now that $\varphi$ is satisfied by a truth function $V$. Define:
- $c(u) = \text{blue}$,
- $c(v) = \text{red}$,
- $c(x_i) = \text{green}$ and $c(\overline{x_i}) = \text{red}$, if $V(x_i) = 1$,
- and $c(x_i) = \text{red}$ and $c(\overline{x_i}) = \text{green}$, if $V(x_i) = 0$. 
For example, if $\varphi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots$, $G_\varphi$ contains the following subgraph $G_1$:

Assume now that $\varphi$ is satisfied by a truth function $V$. Define: $c(u) =$ blue, $c(v) =$ red, $c(x_i) =$ green and $c(\overline{x_i}) =$ red, if $V(x_i) = 1$, and $c(x_i) =$ red and $c(\overline{x_i}) =$ green, if $V(x_i) = 0$.

For example: if $V(x_1) = 1, V(x_2) = 1, V(x_4) = 0, \ldots$,
For example, if $\varphi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots$, $G_\varphi$ contains the following subgraph $G_1$:

Assume now that $\varphi$ is satisfied by a truth function $V$. Define:
$c(u) = \text{blue}$, $c(v) = \text{red}$, $c(x_i) = \text{green}$ and $c(\overline{x_i}) = \text{red}$, if $V(x_i) = 1$, and $c(x_i) = \text{red}$ and $c(\overline{x_i}) = \text{green}$, if $V(x_i) = 0$.

For example: if $V(x_1) = 1, V(x_2) = 1, V(x_4) = 0, \ldots$, $c$ can be extended to a coloring of $G_1 \ldots$
3-COLORABILITY

For example, if $\varphi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots$, $G_\varphi$ contains the following subgraph $G_1$:

Assume now that $\varphi$ is satisfied by a truth function $V$. Define:

$c(u) = \text{blue}$, $c(v) = \text{red}$, $c(x_i) = \text{green}$ and $c(\overline{x_i}) = \text{red}$, if $V(x_i) = 1$,
and $c(x_i) = \text{red}$ and $c(\overline{x_i}) = \text{green}$, if $V(x_i) = 0$.

For $V(x_1) = 1$, $V(x_2) = 1$, $V(x_4) = 1$, $\ldots$, $G_1$ can also be colored $\ldots$
For example, if \( \varphi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots \), \( G_\varphi \) contains the following subgraph \( G_1 \):

Assume now that \( \varphi \) is satisfied by a truth function \( V \). Define:

\[
c(u) = \text{blue}, \quad c(v) = \text{red}, \quad c(x_i) = \text{green} \quad \text{and} \quad c(\overline{x_i}) = \text{red}, \quad \text{if} \quad V(x_i) = 1,
\]

and

\[
c(x_i) = \text{red} \quad \text{and} \quad c(\overline{x_i}) = \text{green}, \quad \text{if} \quad V(x_i) = 0.
\]

... also for \( V(x_1) = 0, V(x_2) = 0, V(x_4) = 0 \) etc.
For example, if $\varphi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots$, $G_\varphi$ contains the following subgraph $G_1$:

For the other direction, assume that $G_\varphi$ has a coloring $c$ (w.l.o.g, $c(u) = \text{blue}$ and $c(v) = \text{red}$). Define $V(x_i) = 1$ if $c(x_i) = \text{green}$, and $V(x_i) = 0$ if $c(x_i) = \text{red}$. 
For example, if $\varphi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots$, $G_\varphi$ contains the following subgraph $G_1$:

For the other direction, assume that $G_\varphi$ has a coloring $c$ (w.l.o.g, $c(u) = \text{blue}$ and $c(v) = \text{red}$). Define $V(x_i) = 1$ if $c(x_i) = \text{green}$, and $V(x_i) = 0$ if $c(x_i) = \text{red}$. This is a truth function $V : \{x_1, \ldots, x_n\} \to \{0, 1\}$, since all $x_i$-nodes are red or green (because $u$ is colored blue).
For example, if \( \varphi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots \), \( G_\varphi \) contains the following subgraph \( G_1 \):

\[ u \]

\[ x_1 \]
\[ \overline{x_1} \]
\[ x_2 \]
\[ \overline{x_2} \]
\[ x_4 \]
\[ \overline{x_4} \]

\[ \ldots \]

\[ z_1 \]
\[ \overline{z_1} \]
\[ v \]
\[ \overline{v} \]
\[ a_1 \]
\[ \overline{a_1} \]
\[ y_1 \]
\[ \overline{y_1} \]
\[ c_1 \]
\[ \overline{c_1} \]

\[ b_1 \]
\[ \overline{b_1} \]

\[ \ldots \]

Assume \( V \) does not satisfy \( C_\varphi \). Then there is clause, say \( C_1 \), with \( V \nmid C_1 \), i.e., all literals in \( C_1 \) are false.
3-Colorability

For example, if $\varphi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots$, $G_\varphi$ contains the following subgraph $G_1$:

\[ \begin{array}{c}
\text{u} \\
\text{x}_1 \quad \text{x}_2 \quad \text{x}_4 \\
\text{\overline{x}_1} \quad \text{\overline{x}_2} \quad \text{\overline{x}_4} \\
\text{\ldots} \\
\text{z}_1 \quad \text{a}_1 \\
\text{v} \\
\text{y}_1 \\
\text{b}_1 \\
\text{c}_1
\end{array} \]

... Assume $V$ does not satisfy $C_\varphi$. Then there is clause, say $C_1$, with $V \not\models C_1$, i.e., all literals in $C_1$ are false.

Then $b_1, c_1$ must be colored blue or green.

If, w.l.o.g., $c(b_1) = \text{green}$ and $c(c_1) = \text{blue}$, \ldots
For example, if \( \varphi = (x_1 \lor \neg x_2 \lor x_4) \land \ldots \), \( G_\varphi \) contains the following subgraph \( G_1 \):

\[
\begin{array}{c}
\bullet u \\
\bullet x_1 \\
\bullet \overline{x_1} \\
\bullet x_2 \\
\bullet \overline{x_2} \\
\bullet \ldots \\
\bullet x_4 \\
\bullet \overline{x_4} \\
\bullet v \\
\bullet y_1 \\
\bullet z_1 \\
\bullet a_1 \\
\bullet b_1 \\
\bullet c_1
\end{array}
\]

\ldots Assume \( V \) does not satisfy \( C_\varphi \). Then there is clause, say \( C_1 \), with \( V \not\models C_1 \), i.e., all literals in \( C_1 \) are false. Then \( b_1, c_1 \) must be colored blue or green.

If, w.l.o.g., \( c(b_1) = \text{green} \) and \( c(c_1) = \text{blue} \), \ldots then \( a_1 \) must be colored red, a contradiction.
**3-Colorability**

**Proof (summary).**

Thus we have constructed a function $f$ that assigns to each 3CNF-SAT formula $\varphi = C_1 \land \cdots \land C_m$ a graph $G_\varphi$ such that

$\varphi$ is satisfiable $\iff$ $G_\varphi$ has a coloring with colors \{red, blue, green\}.

Since the constructed graph $G_\varphi$ has $2n + 5m + 2$ vertices, $f$ can be computed in polynomial time.

**Notice:**

- Actually, what we have proven is:
  $3\text{CNF-SAT} \leq_P k\text{-COLORABILITY}$, for $k \geq 3$.

- The corresponding search problem “Given a graph, find a 3-coloring . . .” is in the complexity class Function NP (FNP).
3-COLORABILITY

Proof (summary).

Thus we have constructed a function $f$ that assigns to each 3CNF-SAT formula $\varphi = C_1 \land \cdots \land C_m$ a graph $G_{\varphi}$ such that

$\varphi$ is satisfiable $\iff$ $G_{\varphi}$ has a coloring with colors \{red, blue, green\}.

Since the constructed graph $G_{\varphi}$ has $2n + 5m + 2$ vertices, $f$ can be computed in polynomial time.

Notice:

- Actually, what we have proven is:
  
  $3\text{CNF-SAT} \leq_p k\text{-COLORABILITY}$, for $k \geq 3$.

- The corresponding search problem “Given a graph, find a 3-coloring . . .” is in the complexity class Function NP (FNP).
Summary

- Short reminder on set-theoretical notions and operations
- Even more operations can be defined for relations
- Distinguish relations (as sets) and relations over variables
- Very basic reminder of graph-theoretical notions
- ... and complexity theory
- Example: $k$-colorability is an NP-complete decision problem
  - ... for $k \geq 3$; for $k = 2$ it is tractable
Summary

- Short reminder on set-theoretical notions and operations
- Even more operations can be defined for relations
- Distinguish relations (as sets) and relations over variables
- Very basic reminder of graph-theoretical notions
- \ldots and complexity theory
- Example: \( k \)-colorability is an NP-complete decision problem
- \ldots for \( k \geq 3 \); for \( k = 2 \) it is tractable
Literature

Rina Dechter.  
Constraint Processing,  
Chapter 1 and 2, Morgan Kaufmann, 2003

Sven Oliver Krumke and Hartmut Noltemeier.  
Graphentheoretische Konzepte und Algorithmen,  
Vieweg+Teubner, 2009

Uwe Schöning.  
Theoretische Informatik – kurzgefasst,  
Spektrum, 2001

Wikipedia contributors,  
Graph theory, Graph (mathematics), Boolean Algebra, Relational Algebra, (2007, April),  