Formale Methoden der Informatik

Block 1: Foundations of Complexity Theory

3. Reductions

Reinhard Pichler

Institut für Informationssysteme
Arbeitsbereich DBAI
Technische Universität Wien

14 October, 2013

Outline

Basic Ideas

Example

Recall the algorithm for solving the **TSP** problem (i.e., what is the length of the shortest tour through n cities?) based on a simple binary search which successively solves instances of the **TSP(D)** problem (i.e., does there exist a tour of length at most B for various values of B?).

Observation

The task of solving the **TSP** problem is thus reduced to the task of solving the **TSP(D)** problem.

Remark. Problem reductions are a key technique in complexity theory. However, they play an important role in any field where problems have to be solved, i.e. the importance of problem reductions is by no means restricted to (theoretical) computer science.

First Idea of Reductions

Idea (not restricted to theoretical computer science)

- Suppose that we want to solve some new problem A (i.e., we want to construct an algorithm for the problem A).
- Further, suppose that we know how to solve some related problem B (i.e., we already have a suitable algorithm for the problem B).
- Idea. Try to solve A by transforming problem A into problem B (i.e., the algorithm for problem A may make use of the algorithm for problem B).

If this problem solving strategy works, we say that problem A is reduced to problem B. $\Longrightarrow$ Problem A is not harder than problem B.

Second Idea of Reductions

Idea (typical for complexity theory)

- Suppose that we know that some problem A is hard to solve (i.e., it has already been proved that there exists no efficient algorithm or no algorithm at all for problem A).
- Further, suppose that we want to prove that some new problem B is also hard (i.e., we suspect that there exists no efficient algorithm or no algorithm at all for problem B).
- Idea. Construct an algorithm for problem A which makes use of the algorithm for problem B).

Conclusion. If such a problem reduction from A to B exists, then problem B must be at least as hard as problem A. $\Longrightarrow$ This problem reduction proves the same hardness result for B as for A.

Reductions and Limiting Resources

Motivation

- The problem reduction from A to B should be easier than the problems involved.
- Otherwise the complexity of problem A is "hidden" in the reduction and a comparison of the complexity of A and B is impossible.
- Conclusion. If problem A can be reduced to problem B then problem A is at most as hard as problem B (or, equivalently, B is at least as hard as A), provided that the problem reduction is sufficiently easy.

Typical requirement in complexity theory

Reductions must be feasible in polynomial time or in logarithmic space.

Turing Reductions

Idea

- The algorithm for problem A uses the algorithm for problem B as a subroutine.
- The algorithm for problem A thus consists of a (polynomial-time or log-space bounded) control program which may call the subroutine for solving instances of problem B arbitrarily often.
- Example. We have already seen in this lecture how to solve the **TSP** problem via binary search and successively calling a subroutine which solves an instance of the **TSP(D)** problem.

Polynomial-time Turing reductions are called Cook reductions.

Many-one Reductions

Idea

- Define a function R from instances of problem A to instances of B, i.e.: each instance x of A is mapped to an instance $R(x)$ of B.
- Solve instance $R(x)$ with the algorithm for problem B. The answer of algorithm B on the instance $R(x)$ is already the correct answer to the instance x of A. We say that x and $R(x)$ are equivalent, e.g. (for decision problems) x is a positive instance of $A \Leftrightarrow R(x)$ is a positive instance of B.
- Remark. Many-one reductions are a special case of Turing reductions, where the subroutine for problem B may be called exactly once and where the computation ends immediately after this call.

Polynomial-time many-one reductions are called Karp reductions.

Some Decision Problems

SAT

INSTANCE: Boolean formula φ.

QUESTION: Is φ satisfiable?

3-SAT

INSTANCE: Boolean formula φ in 3-CNF (i.e., CNF where each clause consists of exactly 3 literals).

QUESTION: Is φ satisfiable?

Some Decision Problems

2-SAT

INSTANCE: Boolean formula φ in 2-CNF (i.e., CNF where each clause consists of exactly 2 literals).

QUESTION: Is φ satisfiable?

INDEPENDENT SET

INSTANCE: Undirected graph $G = (V, E)$ and integer K.

QUESTION: Does there exist an *independent set* I of size $\geq K$?
i.e., $I \subseteq V$, s.t. for all $i, j \in I$ the condition $[i, j] \notin E$ holds?

Example: Turing Reduction

Proposition

There exists a polynomial-time Turing reduction from the **2-SAT** *problem to the* **REACHABILITY** *problem.*

Remark

We have already seen in this lecture that the **REACHABILITY** problem is tractable (i.e., it can be solved in polynomial time).

Conclusion

The **2-SAT** problem is tractable.

Problem Reduction 2-SAT $\rightarrow$ REACHABILITY

Proof sketch

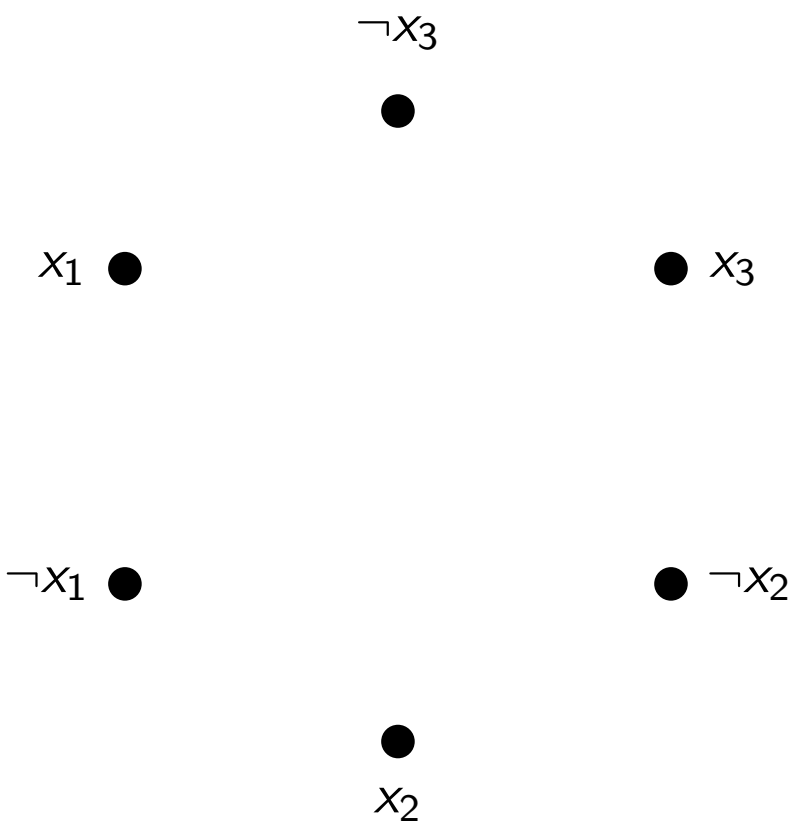
Let φ be an arbitrary instance of **2-SAT**.
Define a graph $G(\varphi)$ as follows:

- The variables of φ and their negations form the vertices of $G(\varphi)$.
- There is an arc (α, β) iff there is a clause $\overline{\alpha} \vee \beta$ or $\beta \vee \overline{\alpha}$ in φ, i.e., if (α, β) is an arc, so is $(\overline{\beta}, \overline{\alpha})$ where $\overline{\alpha}$ is the complement of α.
- Intended meaning of the arcs (α, β): If α is true in some satisfying assignment $\mathcal{I}$ of φ, then β must also be true in $\mathcal{I}$.

It can be shown that φ is unsatisfiable iff there is a variable x such that there are paths from x to $\neg x$ and from $\neg x$ to x in $G(\varphi)$.

Example

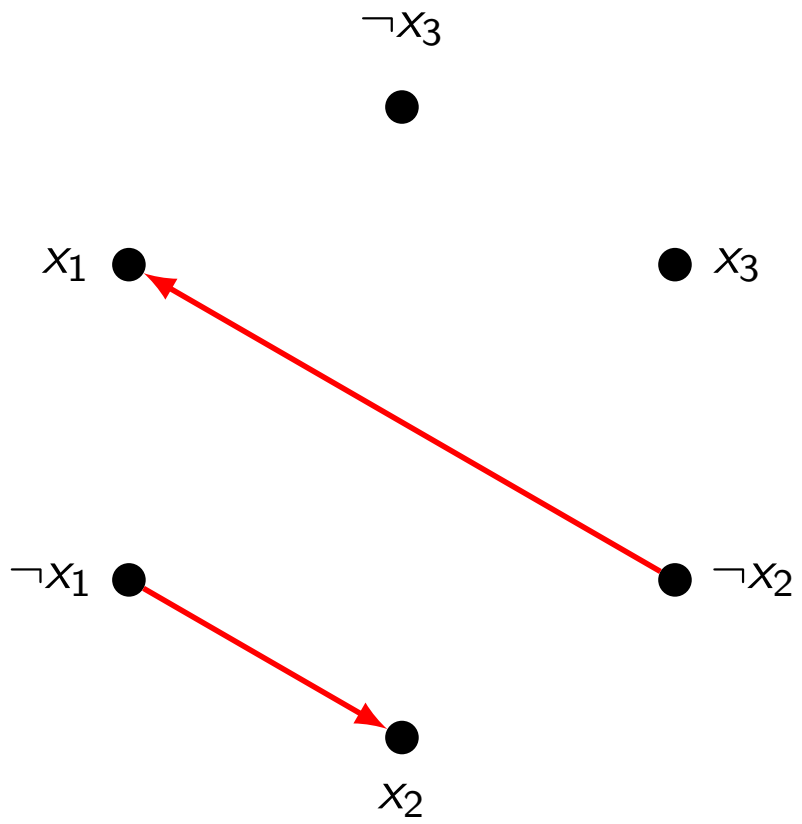
$$\psi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_3)$$



Example

$$\psi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_3)$$

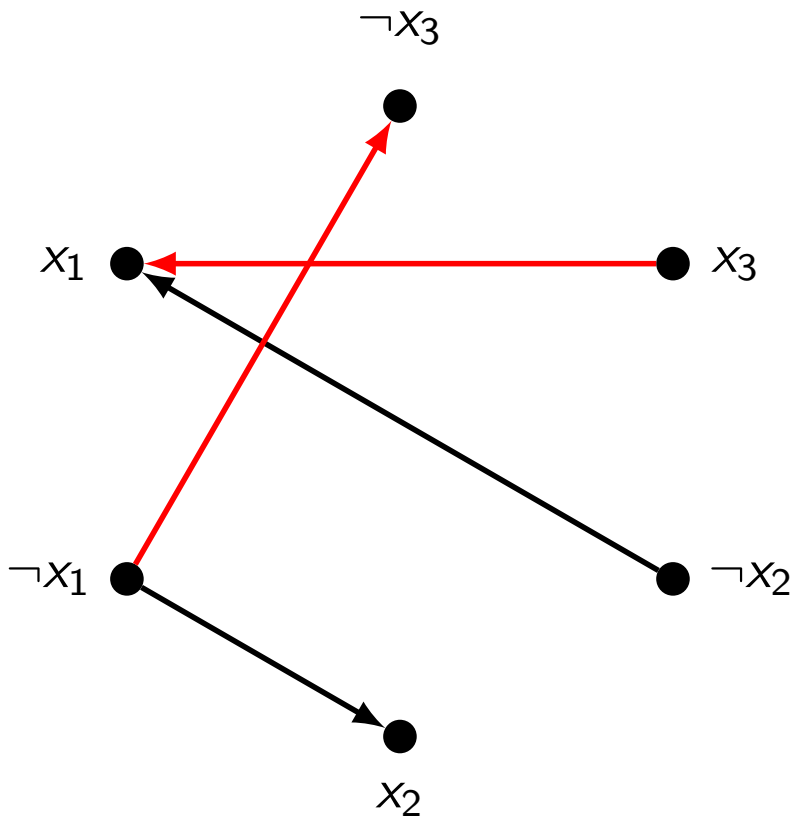
$$\begin{aligned} \neg x_1 &\Rightarrow x_2 \\ \neg x_2 &\Rightarrow x_1 \end{aligned}$$



Example

$$\psi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_3)$$

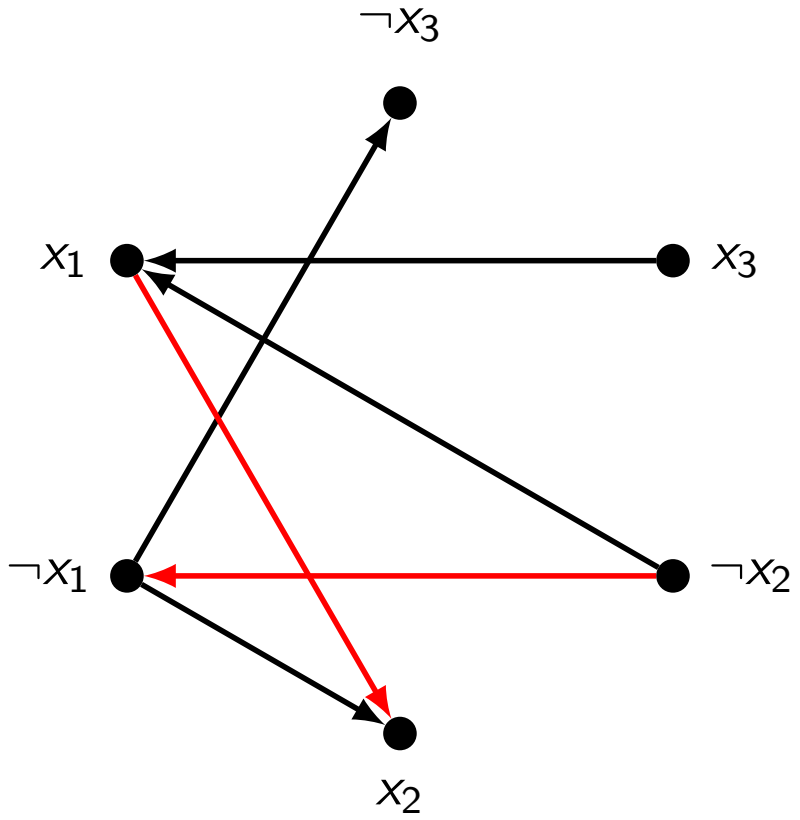
$$\begin{aligned} \neg x_1 &\Rightarrow \neg x_3 \\ x_3 &\Rightarrow x_1 \end{aligned}$$



Example

$$\psi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_3)$$

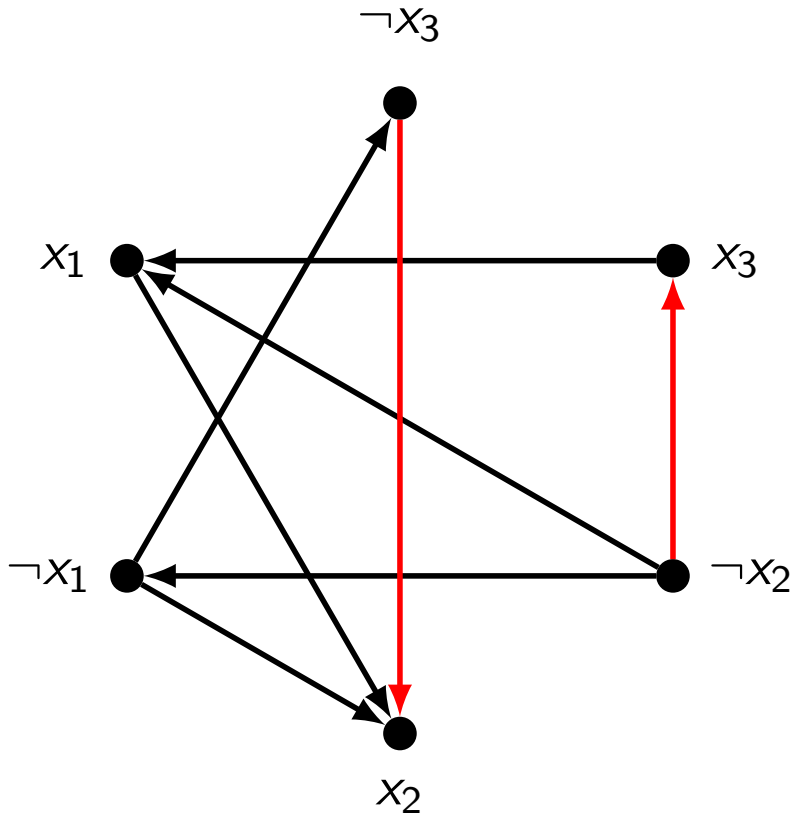
$$\begin{array}{rcl} x_1 & \Rightarrow & x_2 \\ \neg x_2 & \Rightarrow & \neg x_1 \end{array}$$



Example

$$\psi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_3)$$

$$\begin{array}{lcl} \neg x_2 & \Rightarrow & x_3 \\ \neg x_3 & \Rightarrow & x_2 \end{array}$$



Proof sketch (continued)

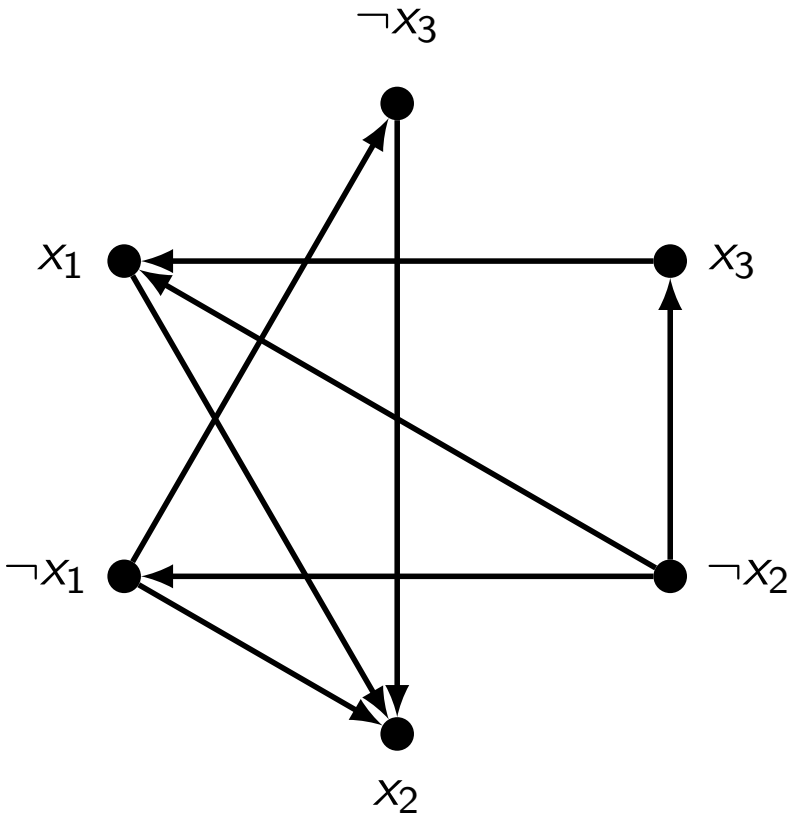
We thus get the following algorithm for **2-SAT**:

1. Construct the graph $G(\varphi)$ as described above;
2. For all variables x of φ check if $\neg x$ is reachable from x in the graph $G(\varphi)$ and x is reachable from $\neg x$;
3. If, by the REACHABILTY tests in step 2, a variable x has been found s.t. both $\neg x$ is reachable from x and x is reachable from $\neg x$, then the **2-SAT**-algorithm returns "no". Otherwise, it returns "yes".

Example (continued)

$$\psi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee x_3)$$

- x_2 reaches no other node
- x_1 reaches only x_2
- $\neg x_3$ reaches only x_2
- $\Rightarrow \psi$ is satisfiable.



Example: Many-one Reduction

Proposition

There exists a polynomial-time many-one reduction from the **3-SAT** *problem to the* **INDEPENDENT SET** *problem.*

Remark

The **SAT** problem is *the* classical NP-complete problem. It remains NP-complete even if we restrict the Boolean formulae to 3-CNF.
(A detailed proof of the NP-completeness of **SAT** and of **3-SAT** will be given in the Komplexitätstheorie lecture in the summer term).

Conclusion

The **INDEPENDENT SET** problem is intractable.

Problem Reduction 3-SAT $\rightarrow$ INDEPENDENT SET

Proof

Let an arbitrary instance of **3-SAT** be given by the Boolean formula φ with m clauses, each consisting of exactly three literals.
Then we construct the following instance of **INDEPENDENT SET**:

The graph G contains a vertex for every literal in φ:
$V = \{l_{11}, l_{12}, l_{13}, \ldots, l_{m1}, l_{m2}, l_{m3}\}$, i.e. $|V| = 3m$.

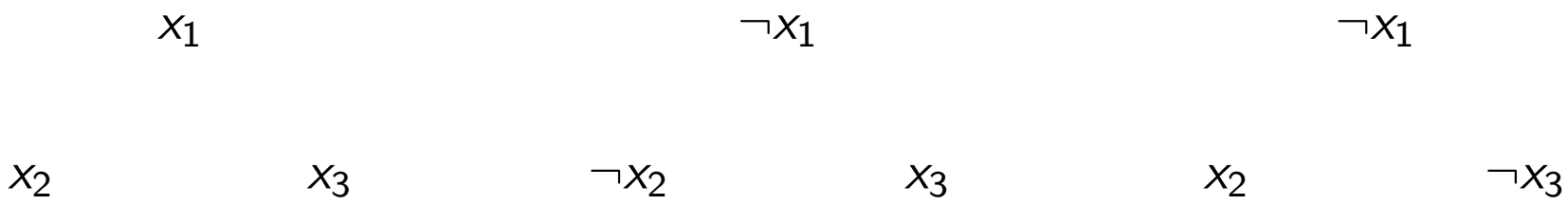
The vertices corresponding to the literals of a clause are all connected by an edge: $E \supseteq \{[l_{i1}, l_{i2}], [l_{i1}, l_{i3}], [l_{i2}, l_{i3}]\}$ for all i, i.e.
G contains a triangle for every clause.

Moreover, for any pair of literals $l_{i\alpha}$, $l_{j\beta}$, if $l_{i\alpha}$ and $l_{j\beta}$ are complementary literals, then E contains an edge $[l_{i\alpha}, l_{j\beta}]$.

Finally, we set $K = m$.

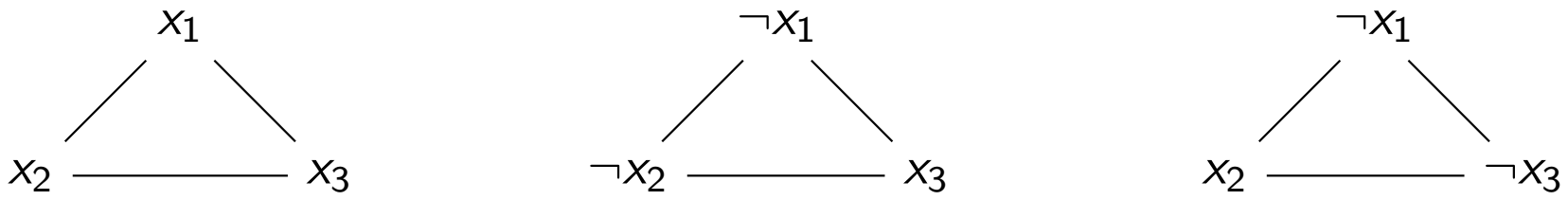
Example

The 3-CNF formula
$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$ is reduced to the following graph:



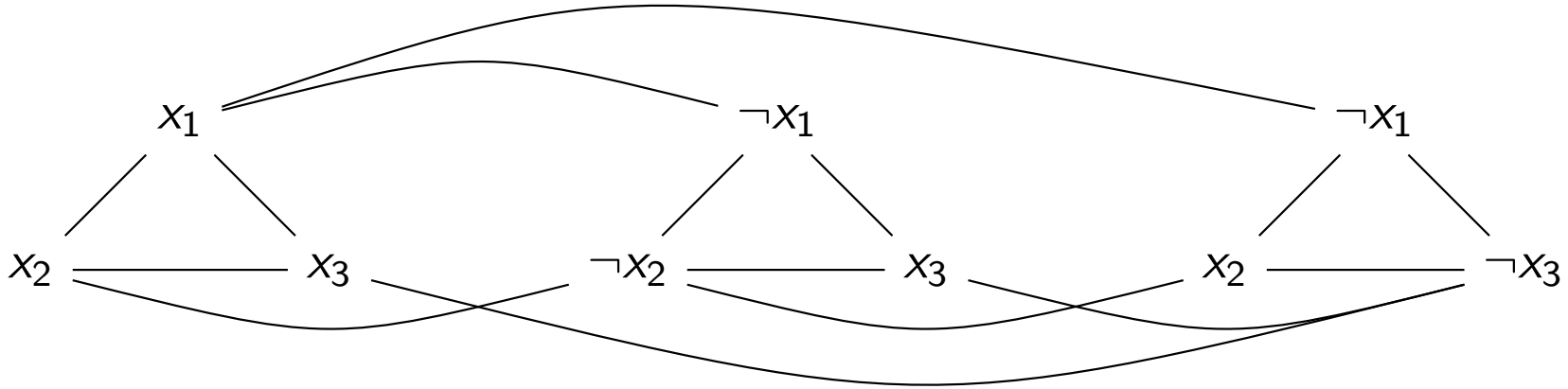
Example

The 3-CNF formula
$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$ is reduced to the following graph:



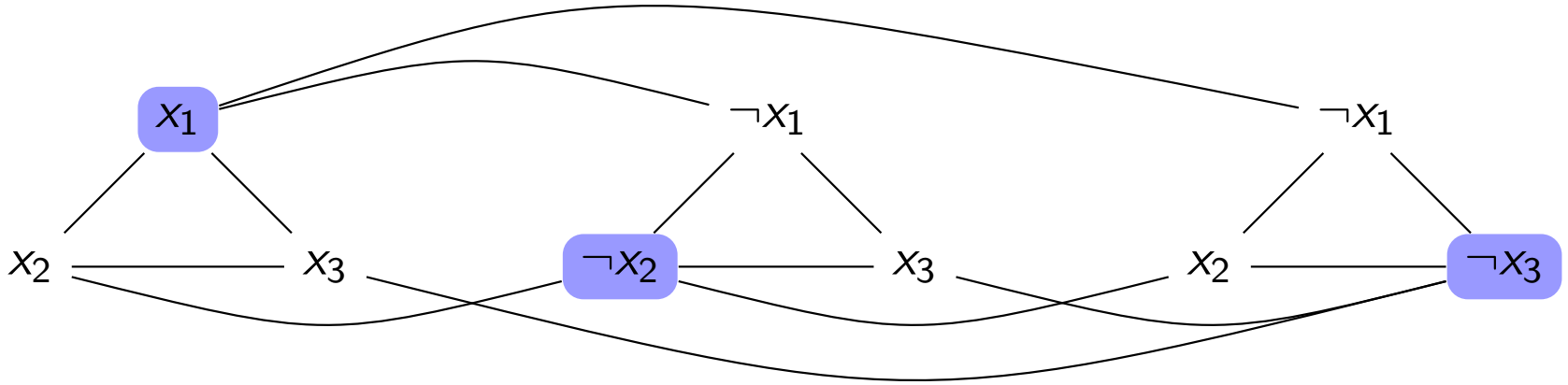
Example

The 3-CNF formula
$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$ is reduced to the following graph:



Example

The 3-CNF formula
$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$ is reduced to the following graph:



The independent set $I = \{I_{11}, I_{22}, I_{33}\}$ corresponds to the satisfying truth assignment T with $T(x_1) = T(\neg x_2) = T(\neg x_3) =$ **true**.

Correctness of the Problem Reduction

Proof (continued)

To prove the correctness of the reduction, we must show that φ is satisfiable $\Leftrightarrow$ the resulting instance (G, K) is positive.

Below, we prove the two directions of this equivalence separately.

Proof of the "$\Leftarrow$" direction

"$\Leftarrow$" Suppose that (G, K) is a positive instance of INDEPENDENT SET, i.e. there exists an independent set I with $|I| \geq K = m$. We have to show that φ is a positive instance of 3-SAT, i.e., φ is satisfiable.

Observe that the vertices in a triangle are all adjacent, and hence I can contain at most one vertex from each triangle. That is, if $\ell_{i\alpha}, \ell_{i\beta} \in I$, then $\alpha = \beta$. Since $|I| \geq K$ and $K = m$, I contains precisely one vertex from each triangle. That is, $I = \{\ell_{1\alpha_1}, \ldots, \ell_{m\alpha_m}\}$ for some combination $\alpha_1, \ldots, \alpha_m$ of values from $\{1, 2, 3\}$.

Proof of the "$\Leftarrow$" direction (continued)

We define the truth assignment T to the variables in φ as follows:

1. A variable x of φ is set to **true** in T if there exists $\ell_{i\alpha} \in I$ s.t. x is the positive literal at position α in the ith clause of φ.
2. The remaining variables are set to **false** in T.

We show next that under T all clauses have a literal that evaluates to **true**, and thus φ is indeed satisfiable.

Consider an arbitrary clause C in φ. Suppose it is the ith clause in φ. Take the literal $\ell_{i\alpha_i}$ in C, i.e., $\ell_{i\alpha_i} \in I$. We claim that $\ell_{i\alpha_i}$ is **true** in T.

Case 1: If $\ell_{i\alpha_i}$ is a positive literal, i.e. some variable y, then $\ell_{i\alpha_i}$ is **true** because y is set to **true** by item (1).

Case 2: Now suppose $\ell_{i\alpha_i}$ is a negative literal of the form $\neg y$. We have to verify that y is set to **false** by T. Suppose this is not the case, i.e. y is set to **true**. Then there exists $\ell_{j\beta} \in I$ s.t. y is the positive literal at position β in the jth clause of φ. Then $\ell_{i\alpha_i}$ and $\ell_{j\beta}$ correspond to complementary literals and thus $[\ell_{i\alpha_i}, \ell_{j\beta}] \in E$. This contradicts the assumption that I is an independent set.

Correctness of the Problem Reduction

Proof of the "⇒" direction

"⇒" Suppose φ is satisfiable. To show that the resulting (G, K) is positive, we have to find an independent set of cardinality at least K.

Since φ is satisfiable, there exists a satisfying truth assignment T of φ. Clearly, T makes **true** at least one literal in each clause of φ. Thus we can define a vertex set $I = \{\ell_{1i_1}, \ldots, \ell_{mi_m}\}$ such that each index i_n is the position of a **true** (in T) literal in the nth clause of φ. (If several literals in a clause are **true** in T, choose one arbitarily). It remains to prove that (i) I is an independent set and (ii) $|I| \geq K$.

The point (ii) is trivial: By the construction of I, we have $|I| = m$, and according to the definition of the reduction we have $K = m$.

The point (i) is also true. Suppose it is false, i.e. there is a pair $\ell_{i\alpha}, \ell_{j\beta} \in I$ with $[\ell_{i\alpha}, \ell_{j\beta}] \in E$. By the construction of G, there are no self-loops in G, i.e. no ℓ_{kl} with $[\ell_{kl}, \ell_{kl}] \in E$. Thus we must have $i \neq j$. Then by the definition of the reduction, $\ell_{i\alpha}$ and $\ell_{j\beta}$ must correspond to complementary literals, of which one must be **false** according to T. Contradiction: to construct I we selected only **true** literals.

Hardness and Completeness

Remarks

- From now on, unless explicitly stated otherwise, we only consider polynomial-time many-one reductions in this lecture.
  We write $\mathcal{P}' \leq \mathcal{P}$ to denote that problem $\mathcal{P}'$ can be reduced to $\mathcal{P}$ (by a polynomial-time many-one reduction).
- The reducibility relation $\leq$ orders problems with respect to their difficulty as it is reflexive and transitive.

Definition

Let C be a complexity class and let $\mathcal{P}$ be a problem.

$\mathcal{P}$ is called C-hard if any problem $\mathcal{P}' \in \mathrm{C}$ is reducible to $\mathcal{P}$.

$\mathcal{P}$ is called C-complete if $\mathcal{P}$ lies in C and $\mathcal{P}$ is C-hard, i.e.:

completeness = membership and hardness

The Role of Completeness in Complexity Theory

- Complete problems are the maximal elements in a class with respect to the reducibility relation $\leq$.
- Complete problems are a central concept and methodological tool in complexity theory:
  - The complexity of a problem is categorized by showing that it is complete for a complexity class.
  - Conversely, complete problems capture the essence of a class.
- Completeness can be used to give negative complexity results:
  A complete problem is the least likely among all problems in C to belong to a weaker class $\mathrm{C}' \subseteq \mathrm{C}$.

The Role of Completeness in Complexity Theory

Proposition

If an NP*-complete problem* $\mathcal{P}$ *is in* P*, then* NP = P*.*

Proof

We know that P $\subseteq$ NP holds. We show that also NP $\subseteq$ P holds:
Let $\mathcal{P}'$ be an arbitrary problem in NP.
As $\mathcal{P}$ is NP-complete, there is a reduction R of $\mathcal{P}'$ to $\mathcal{P}$. But then we can construct a deterministic polynomial-time decision procedure for $\mathcal{P}'$ as follows: For any instance x of problem $\mathcal{P}'$, first compute $R(x)$ and then decide $R(x)$ with a decision procedure for $\mathcal{P}$. Since $\mathcal{P} \in$ P, also $\mathcal{P}' \in$ P.
In total, we thus have NP $\subseteq$ P and, therefore, also NP = P. □

Conclusion. It is generally believed that NP $\neq$ P, in which case there can be no polynomial-time decision procedure for any NP-complete problem!!

Proving Undecidability

Idea

- Recall that the HALTING problem is undecidable.
- There exist many more (in fact, uncountably many) undecidable problems.
- In order to prove the undecidability of some new problem $\mathcal{P}$, we could search for a similarly ingenious proof as the diagonalization.
- Better idea. In order to prove the undecidability of problem $\mathcal{P}$, we show that the HALTING problem can be reduced to $\mathcal{P}$. Clearly, this establishes the undecidability of $\mathcal{P}$ (since, a decision procedure for $\mathcal{P}$ together with the problem reduction from HALTING to $\mathcal{P}$ would immediately yield a a decision procedure for HALTING, which does not exist as we already know).
- The problem reduction need not be bounded in terms of time or space, but must be computable.

Example

CORRECTNESS

INSTANCE: A program Π for a function that takes a string and outputs a string, and a pair of strings I_1, I_2.
QUESTION: Does Π return I_2 when run on input I_1?

A formal proof that CORRECTNESS is undecidable

The proof proceeds by a reduction from HALTING to CORRECTNESS. Let (Π, I) be an arbitrary instance of the HALTING problem, i.e. Π is a program that takes one string as input, and I is an input for Π.

From this, we construct an instance (Π', I_1, I_2) of CORRECTNESS by setting $I_1 = I_2 = I$, and building Π' from Π as follows:

```
String Π′ (String S) {
    call Π(S);
    return S; }
```

A formal proof that CORRECTNESS is undecidable

It remains to show that the following equivalence holds:

$$\Pi \text{ halts on } I \Leftrightarrow \Pi' \text{ returns } I_2 \text{ on } I_1.$$

"$\Rightarrow$" Suppose Π halts on I. Due to the construction of Π', Π' also halts on the input I and returns I as output. Since $I_1 = I_2 = I$ by the definition of the reduction, we have that Π' returns I_2 on I_1.

"$\Leftarrow$" Suppose Π' returns I_2 on I_1. Since $I_1 = I_2 = I$, we have that Π' returns I on I. Since running Π' on I involves running Π on I, we have that Π halts on I.

Discussion

- Reductions can also be used to show that a problem is not semi-decidable.
- Take a problem $\mathcal{P}$ that is not semi-decidable, e.g., the co-problem of HALTING.
- Suppose there is a reduction from $\mathcal{P}$ to a problem $\mathcal{P}'$.
- Then $\mathcal{P}'$ is not semi-decidable either. Indeed, a semi-decision procedure for $\mathcal{P}'$ in combination with the reduction would imply semi-decidability of $\mathcal{P}$.

Example

INCORRECTNESS

INSTANCE: A program Π for a function that takes a string and outputs a string, and a pair of strings I_1, I_2.
QUESTION: Does Π not return I_2 when run on input I_1?

Proof of non semi-decidability of INCORRECTNESS

Observe that the previous reduction from **HALTING** to **CORRECTNESS** is also a reduction from the complement of **HALTING** (i.e. **co-HALTING**) to **INCORRECTNESS**.

Indeed the equivalence "Π halts on $I \Leftrightarrow \Pi'$ returns I_2 on I_1", which we have proved already, implies the equivalence "Π does not halt on $I \Leftrightarrow \Pi'$ does not return I_2 on I_1"

Since **co-HALTING** is not semi-decidable, we have that **INCORRECTNESS** is not semi-decidable.

Learning Objectives

- Two motivations for reducing one problem (or language) to another.
- Two kinds of reductions (Turing, many-one).
- Limiting the resources used by reductions.
- Cook / Karp reductions.
- Proving the correctness of problem reductions.
- The definitions of C-hard and C-complete problems for a complexity class C.
- Understanding the role of complete problems in complexity theory.
- Proving undecidability by reduction from the HALTING problem.
- Proving non-semi-decidability by reduction from the co-HALTING problem.