

CS-E4530

Computational Complexity Theory

Janne H. Korhonen

Aalto University

Spring 2018

Lecture 4:

Reductions

Reductions

- **Recall our previous discussion about reductions**
- **Reduction R from problem L_1 to problem L_2 :**
 - an algorithm that transform an instance x of problem L_1 to an equivalent instance $R(x)$ of problem L_2
- **Relates the complexities of problems L_1 and L_2**
 - Technical requirement: **efficiency**
 - Different notions of reduction
- **This lecture: formalise these notions**

Many-to-one Reductions

Many-to-one Reductions

- **Our basic notion of reduction:**
 - Most reductions we meet on this course are many-to-one reductions
- **Reductions between decision problems L_1 and L_2**
 - Maps instances from L_1 to L_2
 - Preserves yes-instances and no-instances
 - No postprocessing

Many-to-one Reductions: Definitions

Definition

Let $L_1, L_2 \subseteq \{0, 1\}^*$ be languages. A **many-to-one reduction** (often called simply **reduction**) from L_1 to L_2 is a computable function $R: \{0, 1\}^* \rightarrow \{0, 1\}^*$ such that for every $x \in \{0, 1\}^*$

$$x \in L_1 \text{ if and only if } R(x) \in L_2.$$

Definition

If there is a reduction from L_1 to L_2 , we say that L_1 **reduces to** L_2 , and write $L_1 \leq L_2$.

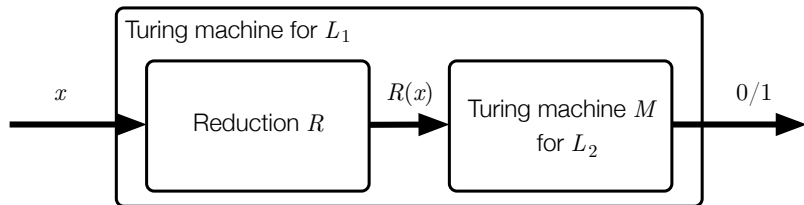
Using Reductions

- **Assume we have:**

- A reduction from R from L_1 to L_2
- A Turing machine M that decides L_2

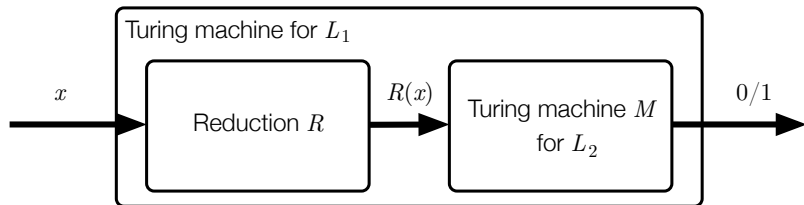
- **Then we can decide L_1 :**

- Transform instance x of L_1 into an instance $R(x)$ of L_2
- Decide $R(x)$ using M



Using Reductions

- **Assume we have:**
 - A reduction from R from L_1 to L_2 running **in time** $T_1(n)$
 - A Turing machine M that decides L_2 **in time** $T_2(n)$
- **Then we can decide** L_1 **in time** $O(T_1(n) + T_2(T_1(n)))$:
 - Transform instance x of L_1 into an instance $R(x)$ of L_2
 - Decide $R(x)$ using M
 - **Note:** $|R(x)| \leq T_1(|x|)$



Polynomial-time Reductions

Definition

Let $L_1, L_2 \subseteq \{0, 1\}^*$ be languages. A **polynomial-time many-to-one reduction** or **Karp reduction** from L_1 to L_2 is a polynomial-time computable function $R: \{0, 1\}^* \rightarrow \{0, 1\}^*$ such that for every $x \in \{0, 1\}^*$

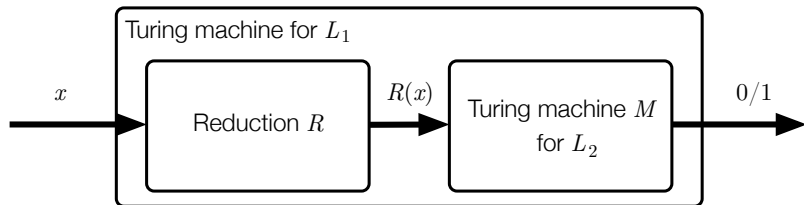
$$x \in L_1 \text{ if and only if } R(x) \in L_2 .$$

Definition

If there is a polynomial-time reduction from L_1 to L_2 , we say that L_1 **reduces to** L_2 in polynomial time, and write $L_1 \leq_p L_2$.

Using Reductions

- **Assume we have:**
 - A **polynomial-time** reduction from R from L_1 to L_2
 - A **polynomial-time** Turing machine M that decides L_2
- **Then we can decide L_1 in polynomial time:**
 - Transform instance x of L_1 into an instance $R(x)$ of L_2
 - Decide $R(x)$ using M
 - $q(p(n))$ is polynomial for polynomials q and p



Transitivity and Reflexivity

Theorem (Transitivity)

Let L_1 , L_2 and L_3 be languages. If $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$, then $L_1 \leq_p L_3$.

- **Proof:** apply reductions sequentially.

Theorem (Transitivity)

Let L be a language. Then $L \leq_p L$

- **Proof:** trivial.
- **Together, these imply that \leq_p is a preorder**

Example: Graph Colouring

Colourings

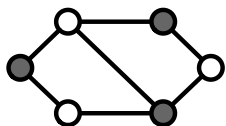
Definition

Let k be a fixed positive integer, and let $G = (V, E)$ be an undirected graph. A k -colouring of G is a function

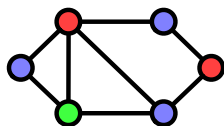
$$c: V \rightarrow \{1, 2, \dots, k\}$$

such that for adjacent vertices v and u , we have $c(v) \neq c(u)$.

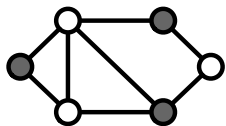
Colourings: Examples



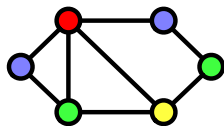
2-colouring



3-colouring



not valid
2-colouring



4-colouring

The k -colouring Problem

k -colouring problem (k -COL)

- **Instance:** Graph $G = (V, E)$.
 - **Question:** Is there a k -colouring of G ?
-
- **We'll use reductions to study relative complexity of the following k -colouring problems:**
 - **2-colouring**
 - **3-colouring**
 - **4-colouring**

2-colouring to 3-colouring

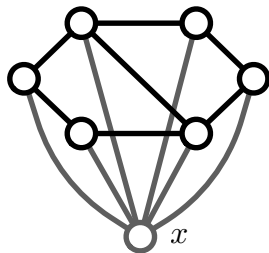
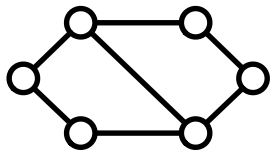
Theorem

There is a polynomial-time reduction from 2-colouring to 3-colouring.

- **We have to show that there is a polynomial time reduction R such that:**
 - R maps any graph G to a new graph $R(G)$
 - If G has a 2-colouring, then $R(G)$ has a 3-colouring
 - If $R(G)$ has a 3-colouring, then G has a 2-colouring

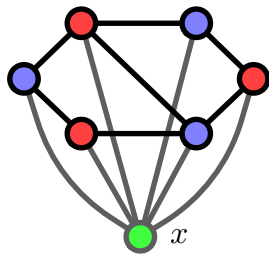
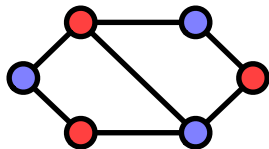
2-colouring to 3-colouring: Proof

- **Given input graph G , construct $R(G)$:**
 - Add a new vertex x to the graph
 - Connect x to all original vertices



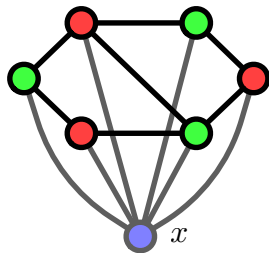
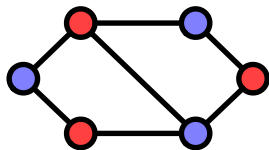
2-colouring to 3-colouring: Proof

- If G has a 2-colouring, then $R(G)$ has 3-colouring:
 - Colour original vertices the same way
 - Colour x with colour 3



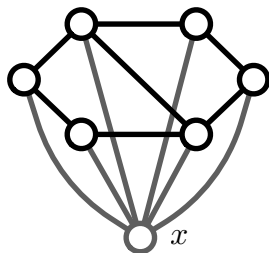
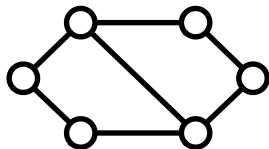
2-colouring to 3-colouring: Proof

- If $R(G)$ has a 3-colouring, then G has 2-colouring:
 - Original vertices cannot use the colour of x
 - Thus, original vertices are coloured with 2 colours
 - This is a 2-colouring of the original graph after renaming the colours



2-colouring to 3-colouring: Proof

- Construction is clearly polynomial-time computable
- **We have:** $2\text{-COL} \leq_p 3\text{-COL}$



3-colouring to 4-colouring

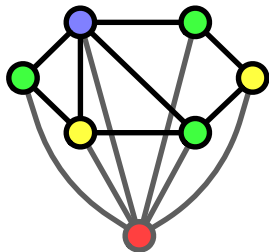
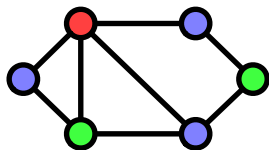
Theorem

The is a polynomial-time reduction from 3-colouring to 4-colouring.

- **We have to show that there is a polynomial time reduction R such that:**
 - R maps any graph G to a new graph $R(G)$
 - If G has a 3-colouring, then $R(G)$ has a 4-colouring
 - If $R(G)$ has a 4-colouring, then G has a 3-colouring
- **Do you immediately see why?**

3-colouring to 4-colouring

- **Same construction works for reduction from 3-colouring to 4-colouring**
 - In fact, from k -colouring to $(k + 1)$ -colouring
 - We have $2\text{-COL} \leq_p 3\text{-COL} \leq_p 4\text{-COL} \leq_p \dots$
 - **What about the other direction?**



4-colouring to 3-colouring

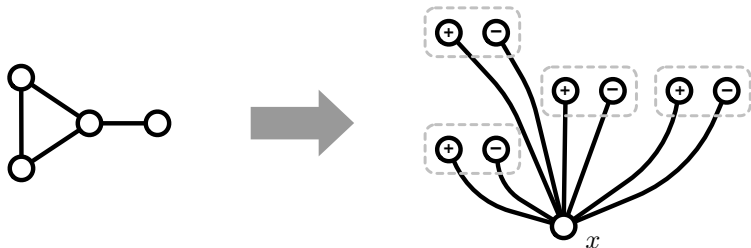
Theorem

The is a polynomial-time reduction from 4-colouring to 3-colouring.

- **We have to show that there is a polynomial time reduction R such that:**
 - R maps any graph G to a new graph $R(G)$
 - If G has a 4-colouring, then $R(G)$ has a 3-colouring
 - If $R(G)$ has a 3-colouring, then G has a 4-colouring
- **This requires considerably more work**

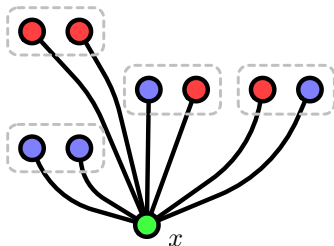
4-colouring to 3-colouring: Proof

- Given input graph G , construct $R(G)$:
 - We start with **a base vertex** x
 - For each original vertex v , add two new vertices v^+ and v^-
 - Connect v^+ and v^- to the vertex x



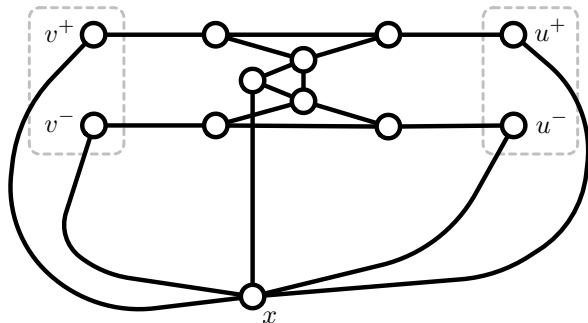
4-colouring to 3-colouring: Proof

- The construction forces all 3-colourings of $R(G)$ to have certain form (if they exist):
 - Vertex x has some colour (say, 3)
 - All vertices v^+, v^- have to use colours 1 and 2
 - **Idea:** use the tuple $(c(v^+), c(v^-))$ to define the colour in the original graph G



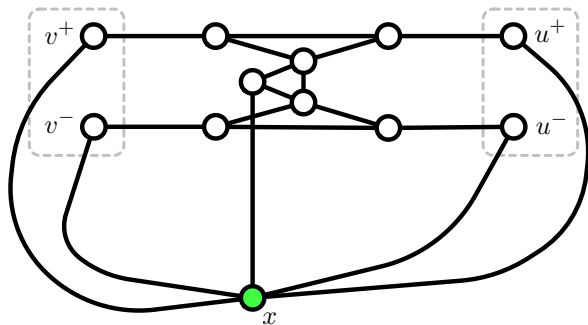
4-colouring to 3-colouring: Proof

- For each edge $(u, v) \in E$ in the original graph G , we add a specific **gadget** to the new graph $R(G)$:
 - Forces $(c(v^+), c(v^-)) \neq (c(u^+), c(u^-))$
 - That is, using $(c(v^+), c(v^-))$ as colour for v in the original graph gives a valid 4-colouring



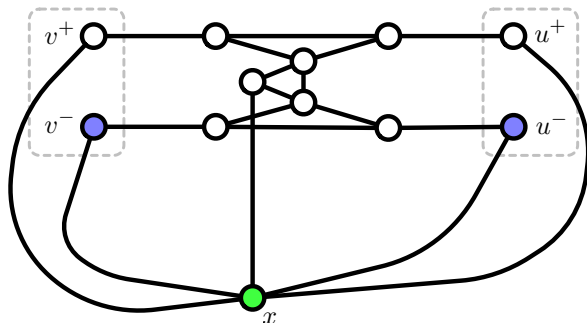
4-colouring to 3-colouring: Proof

- **Construction forces certain colours in the gadget**
 - Assume $c(v^-) = c(u^-)$
 - We show this implies $c(v^+) \neq c(u^+)$



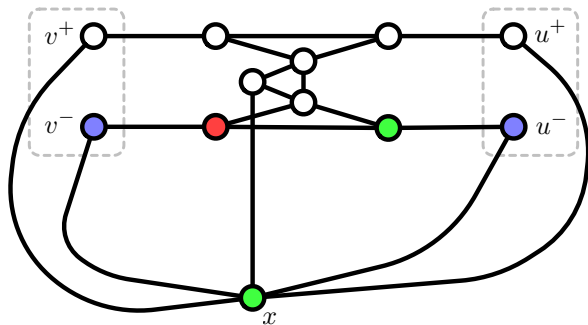
4-colouring to 3-colouring: Proof

- **Construction forces certain colours in the gadget**
 - Assume $c(v^-) = c(u^-)$
 - We show this implies $c(v^+) \neq c(u^+)$



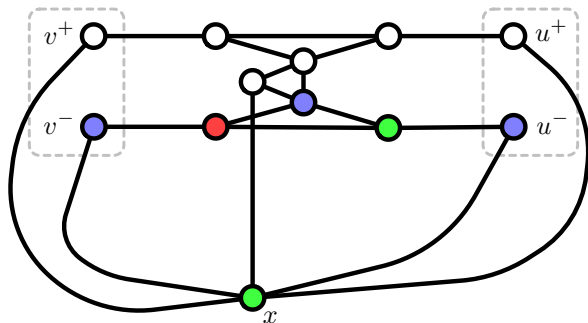
4-colouring to 3-colouring: Proof

- **Construction forces certain colours in the gadget**
 - Assume $c(v^-) = c(u^-)$
 - We show this implies $c(v^+) \neq c(u^+)$



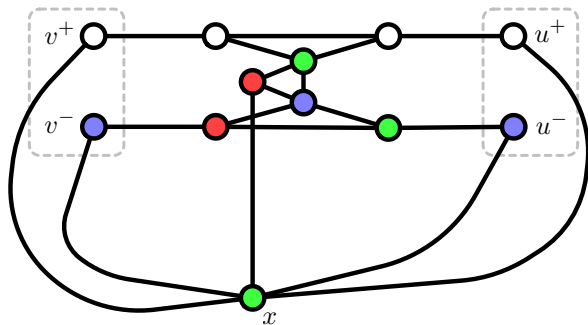
4-colouring to 3-colouring: Proof

- **Construction forces certain colours in the gadget**
 - Assume $c(v^-) = c(u^-)$
 - We show this implies $c(v^+) \neq c(u^+)$



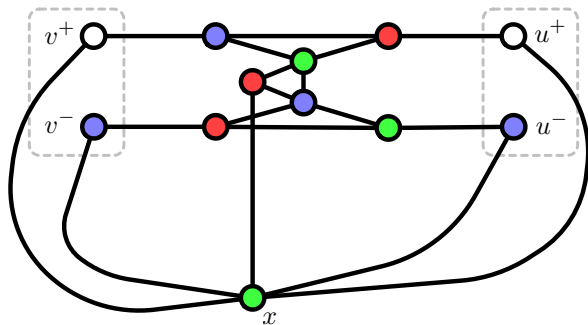
4-colouring to 3-colouring: Proof

- **Construction forces certain colours in the gadget**
 - Assume $c(v^-) = c(u^-)$
 - We show this implies $c(v^+) \neq c(u^+)$



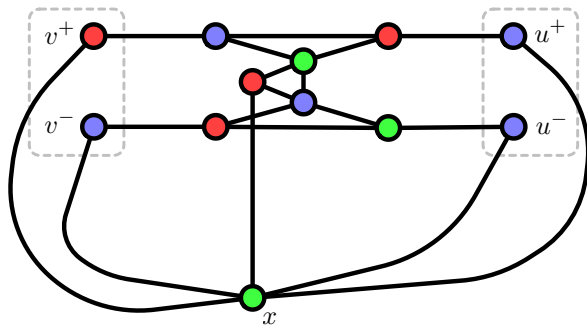
4-colouring to 3-colouring: Proof

- **Construction forces certain colours in the gadget**
 - Assume $c(v^-) = c(u^-)$
 - We show this implies $c(v^+) \neq c(u^+)$



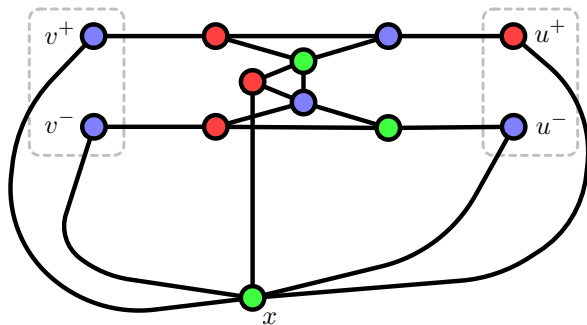
4-colouring to 3-colouring: Proof

- **Construction forces certain colours in the gadget**
 - Assume $c(v^-) = c(u^-)$
 - We show this implies $c(v^+) \neq c(u^+)$



4-colouring to 3-colouring: Proof

- **Construction forces certain colours in the gadget**
 - Assume $c(v^-) = c(u^-)$
 - We show this implies $c(v^+) \neq c(u^+)$



4-colouring to 3-colouring: Proof

- **Similar proof shows that if the original graph G has a 4-colouring, then we can colour $R(G)$ with 3 colours**
 - Reverse of the previous mapping
 - Use two colours for all nodes v^+ and v^-
 - Base vertex x uses the third colour
 - Each gadget can be completed in an obvious way
- **Complexity of the reduction**
 - If G has n vertices and m edges, then $R(G)$ has $1 + 2n + 7m$ vertices
 - Clearly $R(G)$ can be computed in polynomial time

Complexity of Colouring Problems

- **We have proven** $4\text{-COL} \leq_p 3\text{-COL}$
 - Similar reduction works for $k\text{-COL} \leq_p 3\text{-COL}$ for $k \geq 5$
 - All colouring problems are **equally hard** relative to polynomial reductions for $k \geq 3$
- **What about polynomial-time reduction from 3-colouring to 2-colouring?**

Complexity of Colouring Problems

- **We have proven** $4\text{-COL} \leq_p 3\text{-COL}$
 - Similar reduction works for $k\text{-COL} \leq_p 3\text{-COL}$ for $k \geq 5$
 - All colouring problems are **equally hard** relative to polynomial reductions for $k \geq 3$
- **What about polynomial-time reduction from 3-colouring to 2-colouring?**
 - **2-colouring:** in class P
 - **3-colouring:** believed to be not in class P
 - This would imply that the reduction does not exist

Turing Reductions

Turing Reductions

- **More powerful notion of reduction:**
 - Corresponds to **subroutine calls**
 - Turing reduction may make multiple calls to the target language
 - Turing reduction may perform postprocessing

Oracle Turing Machines

- Let $L \subseteq \{0, 1\}^*$ be a language
- An **oracle Turing machine** M with oracle L is a TM:
 - M has a special **oracle tape** (working tape)
 - M has a special state q_{query}
 - When M enters the state q_{query} , and a string $x \in \{0, 1\}^*$ is written on the oracle tape:
 - The head on the oracle tape moves to position 1
 - If $x \in L$, the oracle tape is rewritten with string 1
 - If $x \notin L$, the oracle tape is rewritten with string 0
 - The oracle call counts as **one step** in execution

Turing Reductions

Definition

Let $L_1, L_2 \subseteq \{0, 1\}^*$ be languages. A **Turing reduction** from L_1 to L_2 is an oracle Turing machine with oracle L_2 that decides L_1 .

Definition

Let $L_1, L_2 \subseteq \{0, 1\}^*$ be languages. A **polynomial-time Turing reduction** or **Cook reduction** from L_1 to L_2 is a polynomial-time oracle Turing machine with oracle L_2 that decides L_1 .

Using Turing Reductions

- **Assume we have:**
 - A Turing reduction from R from L_1 to L_2 **in time** $T_1(n)$
 - A Turing machine M that decides L_2 **in time** $T_2(n)$
- **Then we can decide** L_1 **in time** $O(T_1(n) + T_1(n) \cdot T_2(T_1(n)))$:
 - Simulate all oracle calls with M
 - At most $T_1(n)$ calls, each instance at most $T_1(n)$ bits
- **Polynomial** T_1, T_2 **implies polynomial time for** L_1

Closure and Completeness

Closure

Definition

Let R be a class of reductions, and let C be a class of decision problems. We say that C is **closed under R -reductions** if for all languages L_1 and L_2 , having $L_1 \leq_R L_2$ and $L_2 \in C$ implies $L_1 \in C$.

Hardness and Completeness

Definition

Let R be a class of reductions, and let C be a class of decision problems. We say that a language L is **C -hard under R -reductions** if for any language $L' \in C$, there is a R -reduction from L' to L .

Definition

We say that L is **C -complete under R -reductions** if L is C -hard under R -reductions and $L \in C$.

Completeness: Discussion

- **Complete problems are the most difficult problems:**
 - Assume that R -reductions are fast to compute compared to problems in C
 - If there is a fast algorithm for a complete problem, then there is a fast algorithm for **all** problems in C
- **Completeness allows discussion of a class in terms of a single complete problem**
 - Existence of concrete complete problems not obvious
 - We will make this idea more concrete in the next lectures

Reductions: Discussion

Reductions: Discussion

- **Meta-mathematical question: what is the right notion of reduction to use?**
 - Why many-to-one reductions instead of Turing reductions?
 - Why specifically polynomial-time reductions?
- **General rule of thumb: reductions should be easy compared to the complexity class we are studying**
 - Weaker reductions means more fine-grained complexity picture
 - Stronger reductions are easier to work with
 - Use different reductions for studying different classes

Lecture 4: Summary

- Many-to-one reductions
- Completeness and hardness
- (Turing reductions)