Complexity Theory

Lecture 5: Reductions

Tom Gur

http://www.cl.cam.ac.uk/teaching/2324/Complexity

Recap

- Goal: Chart a landscape of complexity classes

Recap

- Goal: Chart a landscape of complexity classes
- $\mathcal{P}$ captures polynomial-time computation

Recap

- Goal: Chart a landscape of complexity classes
- $\mathcal{P}$ captures polynomial-time computation
- $\mathcal{NP}$ captures polynomial-time verification

Recap

- Goal: Chart a landscape of complexity classes
- $\mathcal{P}$ captures polynomial-time computation
- $\mathcal{NP}$ captures polynomial-time verification
- The million dollars question: is $\mathcal{P} \neq \mathcal{NP}$?

Recap

- Goal: Chart a landscape of complexity classes
- $\mathcal{P}$ captures polynomial-time computation
- $\mathcal{NP}$ captures polynomial-time verification
- The million dollars question: is $\mathcal{P} \neq \mathcal{NP}$?
- Natural problems outside of $\mathcal{NP}$?

Recap

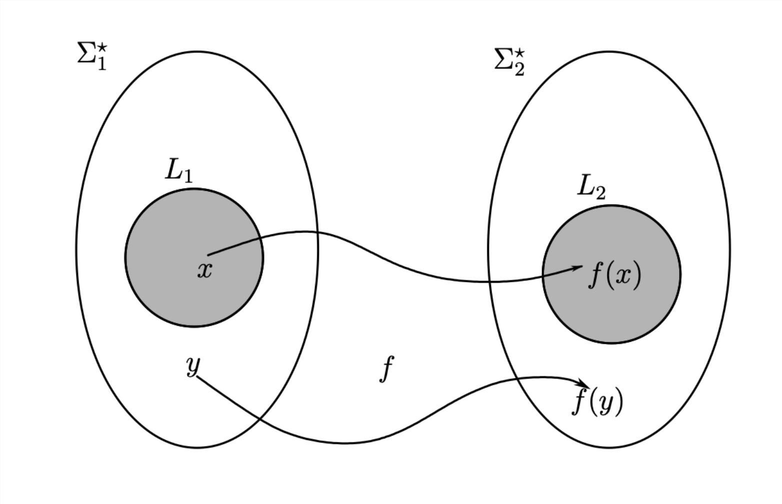
- Goal: Chart a landscape of complexity classes
- $\mathcal{P}$ captures polynomial-time computation
- $\mathcal{NP}$ captures polynomial-time verification
- The million dollars question: is $\mathcal{P} \neq \mathcal{NP}$?
- Natural problems outside of $\mathcal{NP}$?

Recap

- Goal: Chart a landscape of complexity classes
- $\mathcal{P}$ captures polynomial-time computation
- $\mathcal{NP}$ captures polynomial-time verification
- The million dollars question: is $\mathcal{P} \neq \mathcal{NP}$?
- Natural problems outside of $\mathcal{NP}$?

First superpower of complexity theory: solving one problem using another!

Reductions



Reductions

Given two languages $L_1 \subseteq \Sigma_1^\star$, and $L_2 \subseteq \Sigma_2^\star$,

Reductions

Given two languages $L_1 \subseteq \Sigma_1^\star$, and $L_2 \subseteq \Sigma_2^\star$,

A *reduction* of L_1 to L_2 is a *computable* function

$$f : \Sigma_1^\star \to \Sigma_2^\star$$

such that for every string $x \in \Sigma_1^\star$,

$$f(x) \in L_2 \text{ if, and only if, } x \in L_1$$

Reductions

Given two languages $L_1 \subseteq \Sigma_1^\star$, and $L_2 \subseteq \Sigma_2^\star$,

A *reduction* of L_1 to L_2 is a *computable* function

$$f : \Sigma_1^\star \to \Sigma_2^\star$$

such that for every string $x \in \Sigma_1^\star$,

$$f(x) \in L_2 \text{ if, and only if, } x \in L_1$$

What is missing here?

Resource Bounded Reductions

If f is computable by a polynomial time algorithm, we say that L_1 is *polynomial time reducible* to L_2.

$$L_1 \leq_P L_2$$

Resource Bounded Reductions

If f is computable by a polynomial time algorithm, we say that L_1 is *polynomial time reducible* to L_2.

$$L_1 \leq_P L_2$$

If f is also computable in $\mathrm{SPACE}(\log n)$, we write

$$L_1 \leq_L L_2$$

Resource Bounded Reductions

If f is computable by a polynomial time algorithm, we say that L_1 is *polynomial time reducible* to L_2.

$$L_1 \leq_P L_2$$

If f is also computable in $\mathrm{SPACE}(\log n)$, we write

$$L_1 \leq_L L_2$$

Why do we use the $\leq$ notation?

Reductions: an alternative perspective

If $L_1 \leq_P L_2$ we understand that L_1 is no more difficult to solve than L_2, at least as far as polynomial time computation is concerned.

Reductions: an alternative perspective

If $L_1 \leq_P L_2$ we understand that L_1 is no more difficult to solve than L_2, at least as far as polynomial time computation is concerned.

That is to say,

If $L_1 \leq_P L_2$ *and* $L_2 \in \mathsf{P}$, *then* $L_1 \in \mathsf{P}$

Reductions: an alternative perspective

If $L_1 \leq_P L_2$ we understand that L_1 is no more difficult to solve than L_2, at least as far as polynomial time computation is concerned.

That is to say,

If $L_1 \leq_P L_2$ and $L_2 \in \mathsf{P}$, then $L_1 \in \mathsf{P}$

Reductions: an alternative perspective

If $L_1 \leq_P L_2$ we understand that L_1 is no more difficult to solve than L_2, at least as far as polynomial time computation is concerned.

That is to say,

If $L_1 \leq_P L_2$ *and* $L_2 \in \mathsf{P}$*, then* $L_1 \in \mathsf{P}$

We can get an algorithm to decide L_1 by first computing f, and then using the polynomial time algorithm for L_2.

Completeness

Reductions allow us to establish the *relative* complexity of problems, even when we cannot prove absolute lower bounds.

Completeness

Reductions allow us to establish the *relative* complexity of problems, even when we cannot prove absolute lower bounds.

Cook (1972) first showed that there are problems in NP that are maximally difficult.

Completeness

Reductions allow us to establish the *relative* complexity of problems, even when we cannot prove absolute lower bounds.

Cook (1972) first showed that there are problems in NP that are maximally difficult.

A language L is said to be NP-*hard* if for every language $A \in \mathrm{NP}$, $A \leq_P L$.

Completeness

Reductions allow us to establish the *relative* complexity of problems, even when we cannot prove absolute lower bounds.

Cook (1972) first showed that there are problems in NP that are maximally difficult.

A language L is said to be NP-*hard* if for every language $A \in \mathrm{NP}$, $A \leq_P L$.

A language L is NP-*complete* if it is in NP and it is NP-hard.

Completeness

Reductions allow us to establish the *relative* complexity of problems, even when we cannot prove absolute lower bounds.

Cook (1972) first showed that there are problems in NP that are maximally difficult.

A language L is said to be NP-*hard* if for every language $A \in \mathrm{NP}$, $A \leq_P L$.

A language L is NP-*complete* if it is in NP and it is NP-hard.

What languages are NP-complete?

Cook-Levin Theorem: SAT is NP-complete

Cook and Levin independently showed that the language SAT of satisfiable Boolean expressions is NP-complete.

Cook-Levin Theorem: SAT is NP-complete

Cook and Levin independently showed that the language SAT of satisfiable Boolean expressions is NP-complete.

Recall that SAT is in NP. (why?)

Cook-Levin Theorem: SAT is NP-complete

Cook and Levin independently showed that the language SAT of satisfiable Boolean expressions is NP-complete.

Recall that SAT is in NP. (why?)

It remains to show NP-hardness: for every language L in NP, there is a polynomial time reduction from L to SAT. (why is that possible?)

Cook-Levin Theorem: SAT is NP-complete

Cook and Levin independently showed that the language SAT of satisfiable Boolean expressions is NP-complete.

Recall that SAT is in NP. (why?)

It remains to show NP-hardness: for every language L in NP, there is a polynomial time reduction from L to SAT. (why is that possible?)

Since L is in NP, there is a nondeterministic Turing machine

$$M = (Q, \Sigma, s, \delta)$$

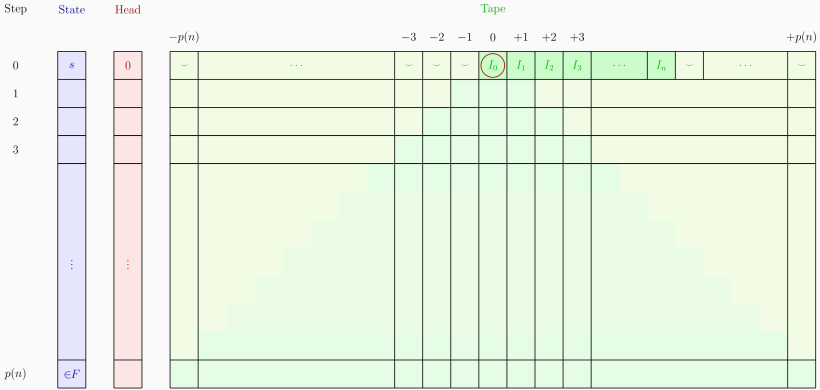
and a bound k such that a string x of length n is in L if, and only if, it is accepted by M within n^k steps.

Turing Machine Tableau

We need to give, for each $x \in \Sigma^\star$, a Boolean expression $f(x)$ which is satisfiable if, and only if, there is an accepting computation of M on input x.

Turing Machine Tableau

We need to give, for each $x \in \Sigma^\star$, a Boolean expression $f(x)$ which is satisfiable if, and only if, there is an accepting computation of M on input x.



$f(x)$ has the following variables:

$$\begin{array}{ll} S_{i,q} & \text{for each } i \leq n^k \text{ and } q \in Q \\ T_{i,j,\sigma} & \text{for each } i,j \leq n^k \text{ and } \sigma \in \Sigma \\ H_{i,j} & \text{for each } i,j \leq n^k \end{array}$$

$f(x)$ has the following variables:

$$\begin{array}{ll} S_{i,q} & \text{for each } i \leq n^k \text{ and } q \in Q \\ T_{i,j,\sigma} & \text{for each } i,j \leq n^k \text{ and } \sigma \in \Sigma \\ H_{i,j} & \text{for each } i,j \leq n^k \end{array}$$

Intuitively, these variables are intended to mean:

- $S_{i,q}$ – the state of the machine at time i is q.

$f(x)$ has the following variables:

$$\begin{array}{ll} S_{i,q} & \text{for each } i \leq n^k \text{ and } q \in Q \\ T_{i,j,\sigma} & \text{for each } i,j \leq n^k \text{ and } \sigma \in \Sigma \\ H_{i,j} & \text{for each } i,j \leq n^k \end{array}$$

Intuitively, these variables are intended to mean:

- $S_{i,q}$ – the state of the machine at time i is q.
- $T_{i,j,\sigma}$ – at time i, the symbol at position j of the tape is σ.

$f(x)$ has the following variables:

$$
\begin{array}{ll}
S_{i,q} & \text{for each } i \leq n^k \text{ and } q \in Q \\
T_{i,j,\sigma} & \text{for each } i,j \leq n^k \text{ and } \sigma \in \Sigma \\
H_{i,j} & \text{for each } i,j \leq n^k
\end{array}
$$

Intuitively, these variables are intended to mean:

- $S_{i,q}$ – the state of the machine at time i is q.
- $T_{i,j,\sigma}$ – at time i, the symbol at position j of the tape is σ.
- $H_{i,j}$ – at time i, the tape head is pointing at tape cell j.

$f(x)$ has the following variables:

$$
\begin{array}{ll}
S_{i,q} & \text{for each } i \leq n^k \text{ and } q \in Q \\
T_{i,j,\sigma} & \text{for each } i, j \leq n^k \text{ and } \sigma \in \Sigma \\
H_{i,j} & \text{for each } i, j \leq n^k
\end{array}
$$

Intuitively, these variables are intended to mean:

- $S_{i,q}$ – the state of the machine at time i is q.
- $T_{i,j,\sigma}$ – at time i, the symbol at position j of the tape is σ.
- $H_{i,j}$ – at time i, the tape head is pointing at tape cell j.

$f(x)$ has the following variables:

$$\begin{array}{ll} S_{i,q} & \text{for each } i \le n^k \text{ and } q \in Q \\ T_{i,j,\sigma} & \text{for each } i,j \le n^k \text{ and } \sigma \in \Sigma \\ H_{i,j} & \text{for each } i,j \le n^k \end{array}$$

Intuitively, these variables are intended to mean:

- $S_{i,q}$ – the state of the machine at time i is q.
- $T_{i,j,\sigma}$ – at time i, the symbol at position j of the tape is σ.
- $H_{i,j}$ – at time i, the tape head is pointing at tape cell j.

We now have to see how to write the formula $f(x)$, so that it enforces these meanings.

Initialization

The initial state is s, and the head is initially at the beginning of the tape.

$$S_{1,s} \wedge H_{1,1}$$

Initialization

The initial state is s, and the head is initially at the beginning of the tape.

$$S_{1,s} \wedge H_{1,1}$$

The initial tape contents are x

$$\bigwedge_{j \leq n} T_{1,j,x_j} \wedge \bigwedge_{n<j} T_{1,j,\sqcup}$$

Consistency

The head is never in two places at once

$$\bigwedge_i \bigwedge_j (H_{i,j} \to \bigwedge_{j' \neq j} (\neg H_{i,j'}))$$

Consistency

The head is never in two places at once

$$\bigwedge_i \bigwedge_j (H_{i,j} \rightarrow \bigwedge_{j' \neq j} (\neg H_{i,j'}))$$

The machine is never in two states at once

$$\bigwedge_q \bigwedge_i (S_{i,q} \rightarrow \bigwedge_{q' \neq q} (\neg S_{i,q'}))$$

Consistency

The head is never in two places at once

$$\bigwedge_i \bigwedge_j (H_{i,j} \rightarrow \bigwedge_{j' \neq j} (\neg H_{i,j'}))$$

The machine is never in two states at once

$$\bigwedge_q \bigwedge_i (S_{i,q} \rightarrow \bigwedge_{q' \neq q} (\neg S_{i,q'}))$$

Each tape cell contains only one symbol

$$\bigwedge_i \bigwedge_j \bigwedge_\sigma (T_{i,j,\sigma} \rightarrow \bigwedge_{\sigma' \neq \sigma} (\neg T_{i,j,\sigma'}))$$

Computation

The tape does not change except under the head

$$\bigwedge_{i} \bigwedge_{j} \bigwedge_{j' \neq j} \bigwedge_{\sigma} (H_{i,j} \wedge T_{i,j',\sigma}) \to T_{i+1,j',\sigma}$$

Computation

The tape does not change except under the head

$$\bigwedge_{i} \bigwedge_{j} \bigwedge_{j' \neq j} \bigwedge_{\sigma} (H_{i,j} \wedge T_{i,j',\sigma}) \to T_{i+1,j',\sigma}$$

Each step is according to δ.

$$\bigwedge_{i} \bigwedge_{j} \bigwedge_{\sigma} \bigwedge_{q} (H_{i,j} \wedge S_{i,q} \wedge T_{i,j,\sigma}) \to \bigvee_{\Delta} (H_{i+1,j'} \wedge S_{i+1,q'} \wedge T_{i+1,j,\sigma'})$$

where Δ is the set of all triples (q', σ', D) such that $((q, \sigma), (q', \sigma', D)) \in \delta$ and

$$j' = \begin{cases} j & \text{if } D = S \\ j - 1 & \text{if } D = L \\ j + 1 & \text{if } D = R \end{cases}$$

where Δ is the set of all triples (q', σ', D) such that $((q, \sigma), (q', \sigma', D)) \in \delta$ and

$$j' = \begin{cases} j & \text{if } D = S \\ j - 1 & \text{if } D = L \\ j + 1 & \text{if } D = R \end{cases}$$

Finally, the accepting state is reached

$$\bigvee_i S_{i,\text{acc}}$$

CNF

A Boolean expression is in *conjunctive normal form* if it is the conjunction of a set of *clauses*, each of which is the disjunction of a set of *literals*, each of these being either a *variable* or the *negation* of a variable.

CNF

A Boolean expression is in *conjunctive normal form* if it is the conjunction of a set of *clauses*, each of which is the disjunction of a set of *literals*, each of these being either a *variable* or the *negation* of a variable.

For any Boolean expression ϕ, there is an equivalent expression ψ in conjunctive normal form.

CNF

A Boolean expression is in *conjunctive normal form* if it is the conjunction of a set of *clauses*, each of which is the disjunction of a set of *literals*, each of these being either a *variable* or the *negation* of a variable.

For any Boolean expression ϕ, there is an equivalent expression ψ in conjunctive normal form.

ψ can be exponentially longer than ϕ.

CNF

A Boolean expression is in *conjunctive normal form* if it is the conjunction of a set of *clauses*, each of which is the disjunction of a set of *literals*, each of these being either a *variable* or the *negation* of a variable.

For any Boolean expression ϕ, there is an equivalent expression ψ in conjunctive normal form.

ψ can be exponentially longer than ϕ.

However, CNF-SAT, the collection of satisfiable CNF expressions, is NP-complete.

3SAT

A Boolean expression is in 3CNF if it is in conjunctive normal form and each clause contains at most 3 literals.

3SAT

A Boolean expression is in 3CNF if it is in conjunctive normal form and each clause contains at most 3 literals.

3SAT is defined as the language consisting of those expressions in 3CNF that are satisfiable.

3SAT

A Boolean expression is in 3CNF if it is in conjunctive normal form and each clause contains at most 3 literals.

3SAT is defined as the language consisting of those expressions in 3CNF that are satisfiable.

3SAT is NP-complete, as there is a polynomial time reduction from CNF-SAT to 3SAT.

Composing Reductions

Polynomial time reductions are clearly closed under composition.

So, if $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then we also have $L_1 \leq_P L_3$.

Composing Reductions

Polynomial time reductions are clearly closed under composition.

So, if $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then we also have $L_1 \leq_P L_3$.

Composing Reductions

Polynomial time reductions are clearly closed under composition.

So, if $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then we also have $L_1 \leq_P L_3$.

If we show, for some problem A in NP that

$$\text{SAT} \leq_P A$$

or

$$\text{3SAT} \leq_P A$$

it follows that A is also NP-complete.

Questions?