CSCI 1590
Intro to Computational Complexity
Diagonalization

John E. Savage

Brown University

February 22, 2008

Summary

The Role of Diagonalization

- A basic challenge of complexity theory is to separate complexity classes. For example, can we show that **P** $\neq$ **NP**? Is **L** $\neq$ **NL**?
- The classes of computable and non-computable languages have been separated using diagonalization.
- Diagonalization enumerates input strings and machines. Languages are created that can't be recognized a) at all or b) within stated resources limits.
- Late we show relativization cannot separate **P** and **NP**.

Diagonalization

Enumeration of Strings and TMs

- Given an alphabet, let the ith string over the alphabet be the ith string in lexicographic order.
- The control unit of a (nondeterministic) TM is an FSM characterized by its next-state, output function $\delta : Q \times \Sigma \mapsto 2^{Q \times \Sigma \times \mu}$ which maps a state and a tape symbol to a set of triples of successor state, symbol, and head movement when reaching the successor state.
  - W.l.o.g. assume each nondeterministic state has $\leq$ two successors.
  - δ can be described by a table with $|Q||\Sigma|$ rows and five columns.
  - The description of a TM, $\lfloor M \rfloor$, is a string describing this table.
  - There is an infinite number of equivalent TMs: Given a TM M with next-state, output function δ, add states that cannot be reached.
- The ith TM is the ith well-formed TM description (table).

Universal Turing Machine

Definition

A universal Turing machine is a DTM that can simulate an arbitrary TM on its input.

A universal TM U simulates an arbitrary TM M (deterministic or nondeterministic) using its description $\lfloor M \rfloor$ and its input string $\mathbf{x}$, written on the tape of U and separated by a special symbol, say $\#$. U maintains the state of M by marking a row in $\lfloor M \rfloor$ as well as the position of M's head on its tape by marking the the tape.

To simulate one computation step by M, U moves its head to the position of M's head, reads the value under it, carries this information back to the marked row in $\lfloor M \rfloor$, and makes a state transition by moving the mark within $\lfloor M \rfloor$. It retains the symbol to be written under M's head as well as the move to be made to the head. It then moves to M's tape, changes the symbol under the head and moves the head.

Diagonalization

Definition

Language L is recursively enumerable (r.e.) if $\exists$ a TM that recognizes L.

Let M_i denote the ith Turing machine. Consider the "diagonal" language.

$$\mathcal{L}_1 = \{\mathbf{w}_i \mid \mathbf{w}_i \text{ not accepted by } M_i\}$$

Theorem

The language $\mathcal{L}_1$ is not is not recursively enumerable.

Proof.

We use proof by contradiction; we assume the existence of a TM M_k that accepts $\mathcal{L}_1$. If $\mathbf{w}_k$ is in $\mathcal{L}_1$, then M_k accepts it, contradicting the definition of $\mathcal{L}_1$. This implies that $\mathbf{w}_k$ is not in $\mathcal{L}_1$. On the other hand, if $\mathbf{w}_k$ is not in $\mathcal{L}_1$, then it is not accepted by M_k. It follows from the definition of $\mathcal{L}_1$ that $\mathbf{w}_k$ is in $\mathcal{L}_1$. Thus, $\mathbf{w}_k$ is in $\mathcal{L}_1 \Leftrightarrow$ it is not in $\mathcal{L}_1$. $\square$

Reducibility

Definition

A language $L_1 \subseteq \Sigma_1^*$ is reducible to language $L_2 \subseteq \Sigma_2^*$ if there is a computable $f : \Sigma_1^* \mapsto \Sigma_2^*$ such that $\mathbf{x} \in L_1 \Leftrightarrow f(\mathbf{x}) \in L_2$.

Definition

A language L is decidable if both L and its complement are recursively enumerable. It is undecidable otherwise.

Lemma

Let L_1 be reducible to L_2. If L_2 is recursively enumerable (r.e.) (decidable), then L_1 is r.e. (decidable). If L_1 is not r.e. (undecidable), L_2 is not r.e. (undecidable).

The Halting Problem is Undecidable

HALT
Instance: $\{\alpha, \mathbf{x}\}$ such that α is a description of Turing machine M_α and $\mathbf{x}$ is an input string to M_α.
Answer: "Yes" if M_α halts on $\mathbf{x}$.

Theorem

HALT *is r.e. but undecidable.*

Proof.

To recognize "Yes" instances of HALT, use a universal TM to simulate M_α on $\mathbf{x}$. To show that this language can't be decided, assume that TM M_H decides it. We show M_H can be used to recognize $\mathcal{L}_1$. On $\{\alpha, \mathbf{x}\}$, M_H determines whether or not M_α halts on $\mathbf{x}$. Build M_1 that invokes M_H on the input. If M_α doesn't halt, M_1 accepts. If M_α halts, M_1 reverses M_H's decision. M_1 recognizes $\mathcal{L}_1$. Contradiction. □

Resource Bounded Reductions

We have seen both computable and polynomial-time reductions between problems. The latter are examples of resource-bounded reductions.

Definition

A language $L_1 \subseteq \Sigma_1^*$ is **R**-reducible to language $L_2 \subseteq \Sigma_2^*$, denoted $L_1 <_R L_2$, if there is a computable $f : \Sigma_1^* \mapsto \Sigma_2^*$ in the class **R** such that $\mathbf{x} \in L_1 \Leftrightarrow f(\mathbf{x}) \in L_2$.

- A reduction is PTIME when $\mathbf{R} = \mathbf{P}$, denoted $L_1 <_p L_2$.
- A reduction is LOGSPACE when $\mathbf{R} = \mathbf{L}$, denoted $L_1 <_{LOG} L_2$.

Compatible and Transitive Reductions

Definition

Let **C** be a complexity class, **R** a class of resource-bounded reductions, and L_1 and L_2 languages. A set of reductions **R** is compatible with **C** if $L_1 <_R L_2$ and L_2 in **C** implies that L_1 is in **C**.

- PTIME reductions ($<_p$) are compatible with **P**.

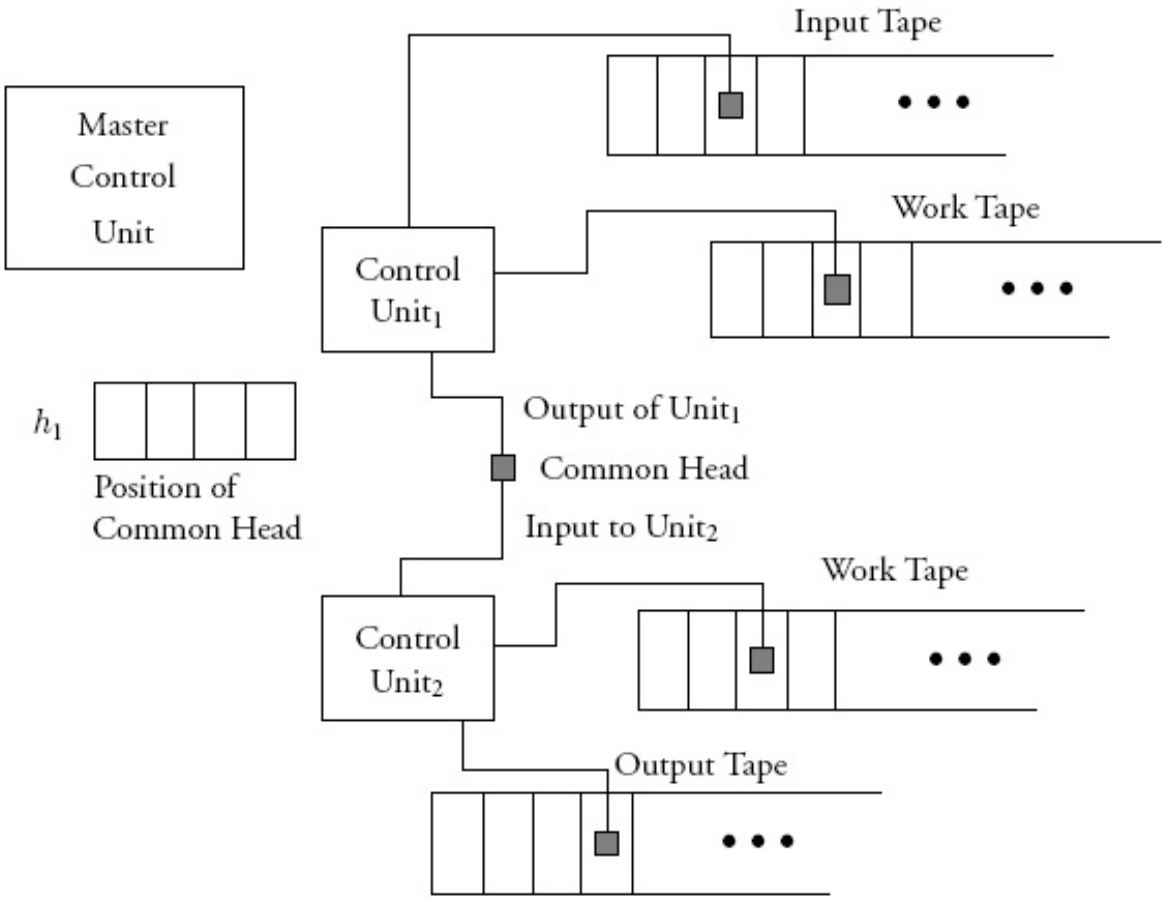
Definition

A class **R** of reductions is transitive if the composition of any two reductions in **R** is another in **R** and for all languages L_1, L_2, and L_3, $L_1 <_R L_2$ and $L_2 <_R L_3$ imply that $L_1 <_R L_3$.

Logspace Reductions

Theorem

Log-space reductions ($<_{LOG}$) are transitive.



Log-space Reductions

Proof.

Let M_1 and M_2 be logspace DTMs computing the two logspace reductions f and g. We show that composition $g(f(\mathbf{x}))$ can be computed in logspace.

Construct DTM M that invokes M_1 and M_2 to compute $g(f(\mathbf{x}))$. M has access to a counter which points to the position in $f(\mathbf{x})$ of its input head. When M_2 needs the next value of $f(\mathbf{x})$, M simulates one more step of M_1. When M_2 needs the preceding value of $f(\mathbf{x})$, M decrements the counter and restarts the simulation of M_1 and computes until the desired output is produced.

Observe that $|f(\mathbf{x})| \leq O(\mathbf{x}^p)$ for $p > 0$ because M_1 uses space $O(\log |\mathbf{x}|)$ and doesn't loop. Thus, M_2 is also uses space logarithmic in $|\mathbf{x}|$. □

Hard and Complete Problems

Definition

Let **R** be a class of reductions, let **C** be a complexity class, and let **R** be compatible with **C**. A language L_2 is hard for **C** under **R**-reductions if for every problem L_1 in **C**, $L_1 <_R L_2$. L_2 is complete for **C** under **R**-reductions if L_2 is also in **C**.

Definition

- Languages in **P** that are hard for **P** under log-space reductions are **P**-complete.
- Languages in **NP** that are hard for **NP** under polynomial-time reductions are **NP**-complete.
- Languages in **PSPACE** that are hard for **PSPACE** under polynomial-time reductions are **PSPACE**-complete.

Hard and Complete Problems

P-complete Problem

CIRCUIT VALUE

Instance: A circuit description with fixed values for its input variables and a designated output gate.

Answer: "Yes" if the output of the circuit has value 1.

LINEAR INEQUALITIES

Instance: An integer-valued $m \times n$ matrix A and column m-vector $\mathbf{b}$.

Answer: "Yes" if there is a rational column n-vector $\mathbf{x} > \mathbf{0}$ (all components are non-negative and at least one is non-zero) such that $A\mathbf{x} \leq \mathbf{b}$.

P-complete problems are hard to parallelize.