Lecture 10

- Relations Between Hilbert's 10th Problem and Gödel's Incompleteness Theorem

Lecture 10

- Relations Between Hilbert's 10th Problem and Gödel's Incompleteness Theorem
- $\mathcal{RE}$-Complete Languages

Lecture 10

- Relations Between Hilbert's 10th Problem and Gödel's Incompleteness Theorem
- $\mathcal{RE}$-Complete Languages
- Description Length/Kolmogorov Complexity

Lecture 10

- Relations Between Hilbert's 10th Problem and Gödel's Incompleteness Theorem
- $\mathcal{RE}$-Complete Languages
- Description Length/Kolmogorov Complexity
- Undecidability of Kolmogorov Complexity

Lecture 10

- Relations Between Hilbert's 10th Problem and Gödel's Incompleteness Theorem
- $\mathcal{RE}$-Complete Languages
- Description Length/Kolmogorov Complexity
- Undecidability of Kolmogorov Complexity
- Turing Reductions

Lecture 10

- Relations Between Hilbert's 10th Problem and Gödel's Incompleteness Theorem
- $\mathcal{RE}$-Complete Languages
- Description Length/Kolmogorov Complexity
- Undecidability of Kolmogorov Complexity
- Turing Reductions
- Proof of Rice Theorem

Lecture 10

- Relations Between Hilbert's 10th Problem and Gödel's Incompleteness Theorem
- $\mathcal{RE}$-Complete Languages
- Description Length/Kolmogorov Complexity
- Undecidability of Kolmogorov Complexity
- Turing Reductions
- Proof of Rice Theorem
- HoHo HiHi

Lecture 10

- Relations Between Hilbert's 10th Problem and Gödel's Incompleteness Theorem
- $\mathcal{RE}$-Complete Languages
- Description Length/Kolmogorov Complexity
- Undecidability of Kolmogorov Complexity
- Turing Reductions
- Proof of Rice Theorem
- HoHo HiHi
- Introduction to Time and Space Complexity

Hilbert's 10th Problem

For every TM, M, it is possible to construct a polynomial in $n+m$ variables,

$$f_M(y_1, y_2, \ldots, y_m, x_1, x_2, \ldots, x_n) \; ,$$

satisfying: for every $w \in \Sigma^*$ there are integers $a_1, a_2, \ldots, a_m$ such that M accepts w iff

$$f_M(a_1, a_2, \ldots, a_m, x_1, x_2, \ldots, x_n)$$

has integer roots $x_1, x_2, \ldots, x_n$.

Hilbert's 10th Problem

Remark: The transformation $\langle M, w \rangle \longrightarrow \langle f_M, a_1, a_2, \ldots, a_m \rangle$ is *computable*.

Hilbert's 10th Problem

Remark: The transformation
$\langle M, w \rangle \longrightarrow \langle f_M, a_1, a_2, \ldots, a_m \rangle$ is *computable*.

After this reduction is established (don't forget it took 70 year to do), it is obvious Hilbert's 10th problem is undecidable.

Number Theory

Number theory can be viewed as the collection of true statements over the model of natural numbers with addition and multiplication, $(\mathcal{N}, +, \cdot)$. For example

- $\forall x \exists y \, [y = x + 1]$ (existence of successor)

Number Theory

Number theory can be viewed as the collection of true statements over the model of natural numbers with addition and multiplication, $(\mathcal{N}, +, \cdot)$. For example

- $\forall x \exists y \, [y = x + 1]$ (existence of successor)
- $\forall x \forall y \forall z \, [x \cdot x \cdot x + y \cdot y \cdot y \neq z \cdot z \cdot z]$
  (Fermat' last theorem for exponent $n = 3$)

Number Theory

Number theory can be viewed as the collection of true statements over the model of natural numbers with addition and multiplication, $(\mathcal{N}, +, \cdot)$. For example

- $\forall x \exists y \, [y = x + 1]$ (existence of successor)
- $\forall x \forall y \forall z \, [x \cdot x \cdot x + y \cdot y \cdot y \neq z \cdot z \cdot z]$
  (Fermat' last theorem for exponent $n = 3$)
- $\forall x \exists p \forall y \forall z \, [\, p > x \bigwedge p \neq (y + 1) \cdot (z + 1) \,]$
  (existence of infinitely many primes)

Number Theory

Number theory can be viewed as the collection of true statements over the model of natural numbers with addition and multiplication, $(\mathcal{N}, +, \cdot)$. For example

- $\forall x \exists y \, [y = x + 1]$ (existence of successor)
- $\forall x \forall y \forall z \, [x \cdot x \cdot x + y \cdot y \cdot y \neq z \cdot z \cdot z]$
  (Fermat' last theorem for exponent $n = 3$)
- $\forall x \exists p \forall y \forall z \, [\, p > x \bigwedge p \neq (y+1) \cdot (z+1) \,]$
  (existence of infinitely many primes)
- $\forall x \exists p \forall y \forall z \, [\, p > x \bigwedge p \neq (y+1) \cdot (z+1) \bigwedge (p+2) \neq (y+1) \cdot (z+1) \,]$
  (the twin prime conjecture)

Number Theory

Number theory can be viewed as the collection of true statements over the model of natural numbers with addition and multiplication, $(\mathcal{N}, +, \cdot)$. For example

- $\forall x \exists y \, [y = x + 1]$ (existence of successor)
- $\forall x \forall y \forall z \, [x \cdot x \cdot x + y \cdot y \cdot y \neq z \cdot z \cdot z]$
  (Fermat' last theorem for exponent $n = 3$)
- $\forall x \exists p \forall y \forall z \, [\, p > x \bigwedge p \neq (y + 1) \cdot (z + 1) \,]$
  (existence of infinitely many primes)
- $\forall x \exists p \forall y \forall z \, [\, p > x \bigwedge p \neq (y + 1) \cdot (z + 1) \bigwedge (p + 2) \neq (y + 1) \cdot (z + 1) \,]$
  (the twin prime conjecture)
- Goldbach conjecture: Every even integer is the sum of two primes. Express it in the language.

Peano Arithmetic

- The theory of numbers is usually denoted $\mathrm{Th}(\mathcal{N}, +, \cdot)$.

Peano Arithmetic

- The theory of numbers is usually denoted $\mathrm{Th}(\mathcal{N}, +, \cdot)$.
- The "usual" system of axioms of number theory is called first-order Peano arithmetic, and denoted by *PA*.

Peano Arithmetic

- The theory of numbers is usually denoted $\mathrm{Th}(\mathcal{N}, +, \cdot)$.
- The “usual” system of axioms of number theory is called first-order Peano arithmetic, and denoted by *PA*.
- *PA* includes axioms about the successor operation, well ordering, commutativity and associativity of $+$ and $\cdot$, distributive law, $\ldots$, and the induction axiom.

Completeness of Logical Theories

A logical theory, Th, with an associated axiom system and a model is called complete if every correct statement is also provable (from the axioms).

Completeness of Logical Theories

A logical theory, Th, with an associated axiom system and a model is called complete if every correct statement is also provable (from the axioms).

Question: Is $\mathrm{Th}(\mathcal{N}, +, \cdot)$ complete?

Gödel's Incompleteness Theorem (1931)

Theorem: $\mathrm{Th}(\mathcal{N}, +, \cdot)$ is incomplete.

Gödel's Incompleteness Theorem (1931)

Theorem: $\mathrm{Th}(\mathcal{N}, +, \cdot)$ is incomplete.

Proof:

- By contradiction, using undecidability of Hilbert's 10th.
- Recall M accepts w iff

$$f_M(a_1, a_2, \ldots, a_m, x_1, x_2, \ldots, x_n)$$

has integer roots $x_1, x_2, \ldots, x_n$.

Gödel's Incompleteness Theorem (1931)

Theorem: $\mathrm{Th}(\mathcal{N}, +, \cdot)$ is incomplete.

Proof:

- By contradiction, using undecidability of Hilbert's 10th.
- Recall M accepts w iff

$$f_M(a_1, a_2, \ldots, a_m, x_1, x_2, \ldots, x_n)$$

  has integer roots $x_1, x_2, \ldots, x_n$.
- Notice that

$$\phi = \exists x_1 \ldots \exists x_n f_M(a_1, \ldots, a_m, x_1, \ldots, x_n) = 0$$

  is a statement in our language.

Gödel's Incompleteness Theorem (1931)

Theorem: $\mathrm{Th}(\mathcal{N}, +, \cdot)$ is incomplete.

Proof:

- By contradiction, using undecidability of Hilbert's 10th.
- Recall M accepts w iff

$$f_M(a_1, a_2, \ldots, a_m, x_1, x_2, \ldots, x_n)$$

  has integer roots $x_1, x_2, \ldots, x_n$.
- Notice that

$$\phi = \exists x_1 \ldots \exists x_n f_M(a_1, \ldots, a_m, x_1, \ldots, x_n) = 0$$

  is a statement in our language.

Gödel's Incompleteness Theorem (1931)

- Either ϕ or $\overline{\phi}$ are correct. If $\mathrm{Th}(\mathcal{N}, +, \cdot)$ was complete, then either ϕ or $\overline{\phi}$ were provable.

Gödel's Incompleteness Theorem (1931)

- Either ϕ or $\overline{\phi}$ are correct. If $\mathrm{Th}(\mathcal{N}, +, \cdot)$ was complete, then either ϕ or $\overline{\phi}$ were provable.
- Can assign one TM to try proving ϕ, another to try prove $\overline{\phi}$.

Gödel's Incompleteness Theorem (1931)

- Either ϕ or $\overline{\phi}$ are correct. If $\text{Th}(\mathcal{N}, +, \cdot)$ was complete, then either ϕ or $\overline{\phi}$ were provable.
- Can assign one TM to try proving ϕ, another to try prove $\overline{\phi}$.
- Exactly one will succeed, determining if M accepts w. Contradiction. ♣

Gödel's Incompleteness Theorem (1931)

- Either ϕ or $\overline{\phi}$ are correct. If $\mathrm{Th}(\mathcal{N}, +, \cdot)$ was complete, then either ϕ or $\overline{\phi}$ were provable.
- Can assign one TM to try proving ϕ, another to try prove $\overline{\phi}$.
- Exactly one will succeed, determining if M accepts w. Contradiction. ♣

Gödel's Incompleteness Theorem (1931)

- Either ϕ or $\overline{\phi}$ are correct. If $\text{Th}(\mathcal{N}, +, \cdot)$ was complete, then either ϕ or $\overline{\phi}$ were provable.
- Can assign one TM to try proving ϕ, another to try prove $\overline{\phi}$.
- Exactly one will succeed, determining if M accepts w. Contradiction. $\clubsuit$

- Important comment: This conceptually simple proof uses the undecidability of Hilbert's 10th, established in 1970. It was not available to Gödel in 1931, when he proved the theorem.

$\mathcal{RE}$-Completeness

Question: Is there a language L that is hardest in the class $\mathcal{RE}$ of enumerable languages (languages accepted by some TM)?

$\mathcal{RE}$-Completeness

Question: Is there a language L that is hardest in the class $\mathcal{RE}$ of enumerable languages (languages accepted by some TM)?

Answer: Well, you have to define what you mean by "hardest language".

$\mathcal{RE}$-Completeness

Question: Is there a language L that is hardest in the class $\mathcal{RE}$ of enumerable languages (languages accepted by some TM)?

Answer: Well, you have to define what you mean by "hardest language".

Definition: A language $L_0 \subseteq \Sigma^*$ is called $\mathcal{RE}$-complete if the following holds

- $L_0 \in \mathcal{RE}$ (membership).
- For every $L \in \mathcal{RE}$, $L \leq_m L_0$ (hardness).

$\mathcal{RE}$-Completeness

Definition: A language $L_0 \subseteq \Sigma^*$ is called $\mathcal{RE}$-complete if the following holds

- $L \in \mathcal{RE}$ (membership).
- For every $L \in \mathcal{RE}$, $L \leq_m L_0$ (hardness).

$\mathcal{RE}$-Completeness

Definition: A language $L_0 \subseteq \Sigma^*$ is called $\mathcal{RE}$-complete if the following holds

- $L \in \mathcal{RE}$ (membership).
- For every $L \in \mathcal{RE}$, $L \leq_m L_0$ (hardness).

The second item means that for every enumerable L there is a mapping reduction f_L from L to L_0. The reduction f_L depends on L and will typically differ from one language to another.

$\mathcal{RE}$-Completeness

Question: Having defined a reasonable notion, we should make sure it is not vacuous, namely verify there is at least one language satisfying it.

$\mathcal{RE}$-Completeness

Question: Having defined a reasonable notion, we should make sure it is not vacuous, namely verify there is at least one language satisfying it.

Theorem: The language A_{TM} is $\mathcal{RE}$-Complete.

$\mathcal{RE}$-Completeness

Question: Having defined a reasonable notion, we should make sure it is not vacuous, namely verify there is at least one language satisfying it.

Theorem: The language A_{TM} is $\mathcal{RE}$-Complete.

Proof:

- The universal machine U accepts the language A_{TM}, so $A_{\text{TM}} \in \mathcal{RE}$.
- Suppose L is in $\mathcal{RE}$, and let M be a TM accepting it. Then $f_L(w) = \langle M, w \rangle$ is a mapping reduction from L to A_{TM} (why?). ♣

Description Length and Information

Consider the two (equal length – 28 bits each) strings

```
0101010101010101010101010101
0010110011101010100110001111
```

Which of these two strings has more information?

Description Length and Information

Consider the two (equal length – 28 bits each) strings

```
0101010101010101010101010101
0010110011101010100110001111
```

Which of these two strings has more information?

This raises the difficult question of what information means, and how can it be measured.

Description Length and Information

Consider the two (equal length – 28 bits each) strings

```
0101010101010101010101010101
0010110011101010100110001111
```

Which of these two strings has more information?

This raises the difficult question of what information means, and how can it be measured.

Following Kolmogorov, we will measure the information of a string by means of its description length.

Information and Description Length

The motivation for Kolmogorov complexity is that phenomena with shorter explanations are typically less complex than phenomena with longer explanations.

Information and Description Length

The motivation for Kolmogorov complexity is that phenomena with shorter explanations are typically less complex than phenomena with longer explanations.

Consequently, we will say that strings with longer description length are more informative than those with shorter description.

Information and Description Length

The motivation for Kolmogorov complexity is that phenomena with shorter explanations are typically less complex than phenomena with longer explanations.

Consequently, we will say that strings with longer description length are more informative than those with shorter description.

Of course, we should still define what description length means.

Information and Description Length

The motivation for Kolmogorov complexity is that phenomena with shorter explanations are typically less complex than phenomena with longer explanations.

Consequently, we will say that strings with longer description length are more informative than those with shorter description.

Of course, we should still define what description length means.

An alternative route (not taken here) is to consider how much a string can be compressed.

Kolmogorov Complexity

In this part of the lecture, we view all TMs as computing functions.

In particular, we can talk about the Universal TM for computing functions.

Kolmogorov Complexity

In this part of the lecture, we view all TMs as computing functions.

In particular, we can talk about the Universal TM for computing functions.

Definition: Let M by a TM, and f_M be the function it computes.
The Kolmogorov Complexity of a string x with respect to M, $K_M(x)$, is defined as the length of the shortest string y satisfying $f_M(y) = x$.

Kolmogorov Complexity

In this part of the lecture, we view all TMs as computing functions.

In particular, we can talk about the Universal TM for computing functions.

Definition: Let M by a TM, and f_M be the function it computes.
The Kolmogorov Complexity of a string x with respect to M, $K_M(x)$, is defined as the length of the shortest string y satisfying $f_M(y) = x$.

If there is no such y, we define $K_M(x) = \infty$.

Kolmogorov Complexity

Hey, this definition is no good. It is totally arbitrary and depends on the particular choice of machine M. Moreover, some strings may have $K_M(x) = \infty$, which is counter intuitive.

Kolmogorov Complexity

Hey, this definition is no good. It is totally arbitrary and depends on the particular choice of machine M.
Moreover, some strings may have $K_M(x) = \infty$, which is counter intuitive.

Well gidday, mates, and no worries . We will immediately show how this can be fixed.

Kolmogorov Complexity

Hey, this definition is no good. It is totally arbitrary and depends on the particular choice of machine M. Moreover, some strings may have $K_M(x) = \infty$, which is counter intuitive.

Well gidday, mates, and no worries . We will immediately show how this can be fixed.

Theorem: Let U be a universal Turing machine. For every Turing machine, M, there is a constant c_M (depending on M alone) such that for every $x \in \Sigma^*$, $K_U(x) \leq K_M(x) + c_M$.

Kolmogorov Complexity

Theorem: Let U be a universal Turing machine. For every Turing machine, M, there is a constant c_M (depending on M alone) such that for every $x \in \Sigma^*$, $K_U(x) \leq K_M(x) + c_M$.

Kolmogorov Complexity

Theorem: Let U be a universal Turing machine. For every Turing machine, M, there is a constant c_M (depending on M alone) such that for every $x \in \Sigma^*$, $K_U(x) \leq K_M(x) + c_M$.

Proof: Let y be a shortest string such that $f_M(y) = x$. Then for the universal TM, U, $f_U(\langle M, y \rangle) = f_M(y) = x$. Using prefix-free encodings for TMs, $\langle M, y \rangle$ is simply the concatenation of $\langle M \rangle$, followed by the string y. So we get

$$K_U(x) \leq |y| + |\langle M \rangle| = K_M(x) + |\langle M \rangle| \; .$$

So the theorem holds where $c_M = |\langle M \rangle|$. ♣.

Kolmogorov Complexity

Corollary: If both U_1 and U_2 are universal Turing machines, then there is a constant c such that for every string x,

$$|K_{U_1}(x) - K_{U_2}(x)| < c\ .$$

Kolmogorov Complexity

Corollary: If both U_1 and U_2 are universal Turing machines, then there is a constant c such that for every string x,

$$|K_{U_1}(x) - K_{U_2}(x)| < c\ .$$

So we can take any universal TM, U, define

$$K(x) = K_U(x)\ ,$$

and refer to this measure as "Kolmogorov complexity of the string x.

Kolmogorov Complexity

Corollary: If both U_1 and U_2 are universal Turing machines, then there is a constant c such that for every string x,

$$|K_{U_1}(x) - K_{U_2}(x)| < c \; .$$

So we can take any universal TM, U, define

$$K(x) = K_U(x) \; ,$$

and refer to this measure as “Kolmogorov complexity of the string x.

We now show that for every string x, $K(x)$ equals at most x’s length plus a constant.

Kolmogorov Complexity

Theorem: There is a constant c such that for every string x, $K(x) \leq |x| + c$.

Pay attention to the order of quantifiers in the statement.

Kolmogorov Complexity

Theorem: There is a constant c such that for every string x, $K(x) \leq |x| + c$.
Pay attention to the order of quantifiers in the statement.

Proof: Let M_{ID} be a TM computing the identity function $f(x) = x$ (*e.g.* a TM that halts immediately). Obviously for any string x, $K_{M_{ID}}(x) = |x|$. By previous theorem, there is a constant c such that for any string x,

$$K(x) = K_U(x) \leq K_{M_{ID}}(x) + c = |x| + c \quad \clubsuit$$

Kolmogorov Complexity

Are there strings whose Kolmogorov complexity is substantially smaller than their own length?

- $K(xx) \leq K(x) + c$

Kolmogorov Complexity

Are there strings whose Kolmogorov complexity is substantially smaller than their own length?

- $K(xx) \leq K(x) + c$
- $K(1^n) \leq \log(n) + c$

Kolmogorov Complexity

Are there strings whose Kolmogorov complexity is substantially smaller than their own length?

- $K(xx) \leq K(x) + c$
- $K(1^n) \leq \log(n) + c$
- $K(1^{2^n}) \leq \log(n) + c$

Kolmogorov Complexity

Are there strings whose Kolmogorov complexity is substantially smaller than their own length?

- $K(xx) \leq K(x) + c$
- $K(1^n) \leq \log(n) + c$
- $K(1^{2^n}) \leq \log(n) + c$

But these strings with very concise description are rare.

Kolmogorov Complexity

A simple counting argument gives

Theorem: For every integer $c \geq 1$, the number of strings in $\{0, 1\}^n$ for which $K(x) \leq n - c$ is at most $2^n/2^{c-1}$.

Kolmogorov Complexity

A simple counting argument gives

Theorem: For every integer $c \geq 1$, the number of strings in $\{0,1\}^n$ for which $K(x) \leq n - c$ is at most $2^n/2^{c-1}$.

Proof: In $\{0,1\}^*$ there is 1 string of length 0, 2 string of length 1, ..., 2^{n-c} string of length $n-c$. The total number of strings up to length $n-c$ is $2^{n+1-c} - 1 < 2^n/2^{c-1}$. So the number of possible descriptions y of length $\leq n-c$ is no more than $2^n/2^{c-1}$. This implies that the number of length n strings whose description length is c shorter than their own length is at most $2^n/2^{c-1}$. ♣

Kolmogorov Complexity Uncomputable

The function $K(\cdot)$ is total (defined for every string x) and unbounded. But is it computable?

Kolmogorov Complexity Uncomputable

The function $K(\cdot)$ is total (defined for every string x) and unbounded. But is it computable?

Theorem: The function $K(\cdot)$ is not computable.

Kolmogorov Complexity Uncomputable

The function $K(\cdot)$ is total (defined for every string x) and unbounded. But is it computable?

Theorem: The function $K(\cdot)$ is not computable.

Proof: By contradiction. For every n let y_n be the lexicographically first string y satisfying $K(y) > n$. Then the sequence $\{y_n\}_{n=1}^{\infty}$ is well defined.

Kolmogorov Complexity Uncomputable

The function $K(\cdot)$ is total (defined for every string x) and unbounded. But is it computable?

Theorem: The function $K(\cdot)$ is not computable.

Proof: By contradiction. For every n let y_n be the lexicographically first string y satisfying $K(y) > n$. Then the sequence $\{y_n\}_{n=1}^{\infty}$ is well defined.

Assume K is computable. We'll show this implies the existance of a constant c such that for every n, $K(y_n) < \log(n) + c$.

Kolmogorov Complexity Uncomputable

Consider the following TM, M: On input n (in binary), M generates, one by one, all binary strings $x_0, x_1, x_2, \ldots$ in lexicographic order. For each x_i it produces, M computes $K(x_i)$.

Kolmogorov Complexity Uncomputable

Consider the following TM, M: On input n (in binary), M generates, one by one, all binary strings $x_0, x_1, x_2, \ldots$ in lexicographic order. For each x_i it produces, M computes $K(x_i)$.

If $K(x_i) > n$, the TM, M, outputs $y = x_i$ and halts. Otherwise, the TM, M, continues to examine the lexicographically next string, x_{i+1}.

Kolmogorov Complexity Uncomputable

Consider the following TM, M: On input n (in binary), M generates, one by one, all binary strings $x_0, x_1, x_2, \ldots$ in lexicographic order. For each x_i it produces, M computes $K(x_i)$.

If $K(x_i) > n$, the TM, M, outputs $y = x_i$ and halts. Otherwise, the TM, M, continues to examine the lexicographically next string, x_{i+1}.

Since the function K is unbounded, it is guaranteed that M will eventually reach a string x satisfying $K(x) > n$.

Kolmogorov Complexity Uncomputable

Conclusion: On input (in binary) n, the TM, M, outputs y_n (the lexicographically first string whose Kolmogorov complexity exceeds n, $K(x) > n$).

Kolmogorov Complexity Uncomputable

Conclusion: On input (in binary) n, the TM, M, outputs y_n (the lexicographically first string whose Kolmogorov complexity exceeds n, $K(x) > n$).

Length of n is $\log_2(n)$. So $K_M(y_n) \leq \log_2(n)$.

Kolmogorov Complexity Uncomputable

Conclusion: On input (in binary) n, the TM, M, outputs y_n (the lexicographically first string whose Kolmogorov complexity exceeds n, $K(x) > n$).

Length of n is $\log_2(n)$. So $K_M(y_n) \leq \log_2(n)$.

We saw that there is a constant c_M such that
for every y, $K(y) \leq K_M(y) + c_M$,
so for every n, $K(y_n) \leq \log_2(n) + c_M$.

Kolmogorov Complexity Uncomputable

We know that there is a constant c_M such that for every y, $K(y) \leq K_M(y) + c_M$, so for every n, $K(y_n) \leq \log_2(n) + c_M$.

Kolmogorov Complexity Uncomputable

We know that there is a constant c_M such that
for every y, $K(y) \leq K_M(y) + c_M$,
so for every n, $K(y_n) \leq \log_2(n) + c_M$.

By definition, for every n, $\quad n < K(y_n)$. Combining the last two inequalities, we get, for every n,

$$n < \log_2(n) + c_M \ .$$

Kolmogorov Complexity Uncomputable

We know that there is a constant c_M such that for every y, $K(y) \leq K_M(y) + c_M$, so for every n, $K(y_n) \leq \log_2(n) + c_M$.

By definition, for every n, $n < K(y_n)$. Combining the last two inequalities, we get, for every n,

$$n < \log_2(n) + c_M \; .$$

But asymptotically n grows faster than $\log_2(n) + c_M$. Contradiction to $K(\cdot)$ computability. ♣

Reducibilities

Notion of reducibility was important for producing a solution to A if we got a solution to B. Inversly, if reducibility from A to B establishes that if A has no solution, neither does B.

- Central working horse was mapping reducibility, $A \leq_m B$.

Reducibilities

Notion of reducibility was important for producing a solution to A if we got a solution to B. Inversly, if reducibility from A to B establishes that if A has no solution, neither does B.

- Central working horse was mapping reducibility, $A \leq_m B$.
- Is mapping reducibility general enough notion to capture above intuition?

Reducibilities

- Is mapping reducibility general enough notion to capture above intuition?

Reducibilities

- Is mapping reducibility general enough notion to capture above intuition?
- Not really. For example, any language L is intuitively reducible to its complement, $\overline{L}$.

Reducibilities

- Is mapping reducibility general enough notion to capture above intuition?
- Not really. For example, any language L is intuitively reducible to its complement, $\overline{L}$.
- An answer to "is $x \in L$" is obtained from an answer to "is $x \in \overline{L}$" by simply reversing the original answer.

Reducibilities

- Is mapping reducibility general enough notion to capture above intuition?
- Not really. For example, any language L is intuitively reducible to its complement, $\overline{L}$.
- An answer to "is $x \in L$" is obtained from an answer to "is $x \in \overline{L}$" by simply reversing the original answer.
- So, in particular, $\overline{A_{\text{TM}}}$ should be reducible to A_{TM}. However certainly $\overline{A_{\text{TM}}} \not\leq_m A_{\text{TM}}$ (why?).

Reducibilities

- Is mapping reducibility general enough notion to capture above intuition?
- Not really. For example, any language L is intuitively reducible to its complement, $\overline{L}$.
- An answer to "is $x \in L$" is obtained from an answer to "is $x \in \overline{L}$" by simply reversing the original answer.
- So, in particular, $\overline{A_{\mathrm{TM}}}$ should be reducible to A_{TM}. However certainly $\overline{A_{\mathrm{TM}}} \not\leq_m A_{\mathrm{TM}}$ (why?).
- We now seek a more general notion of reducibilities than $\leq_m$.

Oracles

Definition: An oracle for a language B is a auxiliary device with two tapes, one called the query tape, the other called the response tape.

- When a string $x \in \Sigma^*$ is written on the query tape, the oracle writes a "yes/no" answer on the response tape.
- If $x \in B$ the oracle writes "yes", while if $x \notin B$ the oracle writes "no".

Oracles

Definition: An oracle for a language B is a auxiliary device with two tapes, one called the query tape, the other called the response tape.

- When a string $x \in \Sigma^*$ is written on the query tape, the oracle writes a "yes/no" answer on the response tape.
- If $x \in B$ the oracle writes "yes", while if $x \notin B$ the oracle writes "no".

Remarks

- The oracle always answers correctly.
- Oracles are not realistic computing devices.

Oracle Turing Machines

Definition: An oracle Turing machine is a TM with access to an oracle.

- The TM can query the oracle, and base its future steps upon the oracle's responses.
- We write M^B to denote a TM with an access to an oracle for the language B.

Oracle Turing Machines

Definition: An oracle Turing machine is a TM with access to an oracle.

- The TM can query the oracle, and base its future steps upon the oracle's responses.
- We write M^B to denote a TM with an access to an oracle for the language B.

Remarks:

- At any step in its computation, M^B can query the oracle on just one string.
- So in a terminating computation only finitely many queries can be made.

Turing Reducibility

Definition: Let A and B be two languages. We say that A is Turing reducible to B and denote $A \leq_T B$, if there is an oracle Turing machine M^B that decides A.

Turing Reducibility

Definition: Let A and B be two languages. We say that A is Turing reducible to B and denote $A \leq_T B$, if there is an oracle Turing machine M^B that decides A.

Theorem: If $A \leq_T B$ and B is decidable, then A is decidable.

Turing Reducibility

Definition: Let A and B be two languages. We say that A is Turing reducible to B and denote $A \leq_T B$, if there is an oracle Turing machine M^B that decides A.

Theorem: If $A \leq_T B$ and B is decidable, then A is decidable.

Simple Observation: If $A \leq_m B$ then $A \leq_T B$. The opposite does not hold.

Rice's Theorem

Theorem: Let $\mathcal{C}$ be a proper non-empty subset of the set of enumerable languages. Denote by $L_{\mathcal{C}}$ the set of all TMs encodings, $\langle M \rangle$, such that $L(M)$ is in $\mathcal{C}$. Then $L_{\mathcal{C}}$ is undecidable.

Rice's Theorem

Theorem: Let $\mathcal{C}$ be a proper non-empty subset of the set of enumerable languages. Denote by $L_{\mathcal{C}}$ the set of all TMs encodings, $\langle M \rangle$, such that $L(M)$ is in $\mathcal{C}$. Then $L_{\mathcal{C}}$ is undecidable.

Proof by reduction from A_{TM}.

Rice's Theorem

Theorem: Let $\mathcal{C}$ be a proper non-empty subset of the set of enumerable languages. Denote by $L_{\mathcal{C}}$ the set of all TMs encodings, $\langle M \rangle$, such that $L(M)$ is in $\mathcal{C}$. Then $L_{\mathcal{C}}$ is undecidable.

Proof by reduction from A_{TM}.

Given M and x, we will construct M_0 such that:

- If M accepts x, then $\langle M_0 \rangle \in L_{\mathcal{C}}$.
- If M does not accept x, then $\langle M_0 \rangle \notin L_{\mathcal{C}}$.

Rice's Theorem

- Without loss of generality, $\emptyset \notin \mathcal{C}$.

Rice's Theorem

- Without loss of generality, $\emptyset \notin \mathcal{C}$.
- Otherwise, look at $\overline{\mathcal{C}}$, also proper and non-empty.

Rice's Theorem

- Without loss of generality, $\emptyset \notin \mathcal{C}$.
- Otherwise, look at $\overline{\mathcal{C}}$, also proper and non-empty.
- Since $\mathcal{C}$ is not empty, there exists some language $L \in \mathcal{C}$. Let M_L be a TM accepting this language (recall $\mathcal{C}$ contains only enumerable languages).

Rice's Theorem

- Without loss of generality, $\emptyset \notin \mathcal{C}$.
- Otherwise, look at $\overline{\mathcal{C}}$, also proper and non-empty.
- Since $\mathcal{C}$ is not empty, there exists some language $L \in \mathcal{C}$. Let M_L be a TM accepting this language (recall $\mathcal{C}$ contains only enumerable languages).
- continued . . .

Rice's Theorem

Given $\langle M, x\rangle$, construct M_0 such that:

- If M accepts x, then $L(M_0) = L \in \mathcal{C}$.

Rice's Theorem

Given $\langle M, x\rangle$, construct M_0 such that:

- If M accepts x, then $L(M_0) = L \in \mathcal{C}$.
- If M does not accept x, then $L(M_0) = \emptyset \notin \mathcal{C}$.

Rice's Theorem

Given $\langle M, x \rangle$, construct M_0 such that:

- If M accepts x, then $L(M_0) = L \in \mathcal{C}$.
- If M does not accept x, then $L(M_0) = \emptyset \notin \mathcal{C}$.

M_0 on input y:

1. Run M on x.
2. If M accepts x, run M_L on y.
   a. if M_L accepts, accept, and
   b. if M_L rejects, reject.

Rice's Theorem

Given $\langle M, x \rangle$, construct M_0 such that:

- If M accepts x, then $L(M_0) = L \in \mathcal{C}$.
- If M does not accept x, then $L(M_0) = \emptyset \notin \mathcal{C}$.

M_0 on input y:

1. Run M on x.
2. If M accepts x, run M_L on y.
   a. if M_L accepts, accept, and
   b. if M_L rejects, reject.

Claim: The transformation $\langle M, x \rangle \rightarrow \langle M_0 \rangle$ is a mapping reduction from A_{TM} to $L_{\mathcal{C}}$.

Rice's Theorem

Proof: M_0 on input y:

1. Run M on x.
2. If M accepts, run M_L on y.
   a. if M_L accepts, accept, and
   b. if M_L rejects, reject.

Rice's Theorem

Proof: M_0 on input y:

1. Run M on x.
2. If M accepts, run M_L on y.
   a. if M_L accepts, accept, and
   b. if M_L rejects, reject.

- The machine M_0 is simply a concatanation of two known TMs – the universal machine, and M_L.

Rice's Theorem

Proof: M_0 on input y:

1. Run M on x.
2. If M accepts, run M_L on y.
   a. if M_L accepts, accept, and
   b. if M_L rejects, reject.

- The machine M_0 is simply a concatanation of two known TMs – the universal machine, and M_L.
- Therefore the transformation $\langle M, x\rangle \to \langle M_0\rangle$ is a computable function, defined for all strings in Σ^*.

Rice's Theorem

Proof: M_0 on input y:

1. Run M on x.
2. If M accepts, run M_L on y.
   a. if M_L accepts, accept, and
   b. if M_L rejects, reject.

- The machine M_0 is simply a concatanation of two known TMs – the universal machine, and M_L.
- Therefore the transformation $\langle M, x\rangle \rightarrow \langle M_0\rangle$ is a computable function, defined for all strings in Σ^*.
- (But what do we actually do with strings not of the form $\langle M, x\rangle$?)

Rice's Proof (Concluded)

- If $\langle M, x\rangle \in A_{\mathrm{TM}}$ then M_0 gets to step 2, and runs M_L on y.

Rice’s Proof (Concluded)

- If $\langle M, x\rangle \in A_{\text{TM}}$ then M_0 gets to step 2, and runs M_L on y.
- In this case, $L(M_0) = L$, so $L(M_0) \in \mathcal{C}$.

Rice's Proof (Concluded)

- If $\langle M, x\rangle \in A_{\text{TM}}$ then M_0 gets to step 2, and runs M_L on y.
- In this case, $L(M_0) = L$, so $L(M_0) \in \mathcal{C}$.
- On the other hand, if $\langle M, x\rangle \notin A_{\text{TM}}$ then M_0 never gets to step 2.

Rice's Proof (Concluded)

- If $\langle M, x\rangle \in A_{\text{TM}}$ then M_0 gets to step 2, and runs M_L on y.
- In this case, $L(M_0) = L$, so $L(M_0) \in \mathcal{C}$.
- On the other hand, if $\langle M, x\rangle \notin A_{\text{TM}}$ then M_0 never gets to step 2.
- In this case, $L(M_0) = \emptyset$, so $L(M_0) \notin \mathcal{C}$.

Rice's Proof (Concluded)

- If $\langle M, x \rangle \in A_{\text{TM}}$ then M_0 gets to step 2, and runs M_L on y.
- In this case, $L(M_0) = L$, so $L(M_0) \in \mathcal{C}$.
- On the other hand, if $\langle M, x \rangle \notin A_{\text{TM}}$ then M_0 never gets to step 2.
- In this case, $L(M_0) = \emptyset$, so $L(M_0) \notin \mathcal{C}$.
- This establishes the fact that $\langle M, x \rangle \in A_{\text{TM}}$ iff $\langle M_0 \rangle \in L_{\mathcal{C}}$. So we have $A_{\text{TM}} \leq_m L_{\mathcal{C}}$, thus $L_{\mathcal{C}}$ is undecidable. ♣

Rice’s Theorem (Reflections)

- Rice’s theorem can be used to show undecidability of properties like

Rice's Theorem (Reflections)

- Rice's theorem can be used to show undecidability of properties like
  - "does $L(M)$ contain infinitely many primes"

Rice’s Theorem (Reflections)

- Rice’s theorem can be used to show undecidability of properties like
  - “does $L(M)$ contain infinitely many primes”
  - “does $L(M)$ contain an arythmetic progression of length 15”

Rice’s Theorem (Reflections)

- Rice’s theorem can be used to show undecidability of properties like
  - “does $L(M)$ contain infinitely many primes”
  - “does $L(M)$ contain an arythmetic progression of length 15”
  - “is $L(M)$ empty”

Rice's Theorem (Reflections)

- Rice's theorem can be used to show undecidability of properties like
  - "does $L(M)$ contain infinitely many primes"
  - "does $L(M)$ contain an arythmetic progression of length 15"
  - "is $L(M)$ empty"
- Decidability of properties related to the encoding itself cannot be inferred from Rice. For example "does $\langle M \rangle$ has an even number of states" is decidable.

Rice’s Theorem (Reflections)

- Rice’s theorem can be used to show undecidability of properties like
  - “does $L(M)$ contain infinitely many primes”
  - “does $L(M)$ contain an arythmetic progression of length 15”
  - “is $L(M)$ empty”
- Decidability of properties related to the encoding itself cannot be inferred from Rice. For example “does $\langle M \rangle$ has an even number of states” is decidable.
- Properties like “does M reaches state q_6 on the empty input string” are undecidable, but this does not follow from Rice’s theorem.

Levitation without Meditation

Complexity

A decidable problem is

- computationally solvable in principle,
- but not necessarily in practice.

Problem is resource consumption:

- time
- space

Example

Consider

$$A = \{0^n 1^n | n \geq 0\}$$

Clearly this language is decidable.

Question: How much time does a single-tape TM need to decide it?

Example

M_1: On input w where w is a string,

- Scan across tape and *reject* if 0 is found to the right of a 1.
- Repeat the following if both 0s and 1s appear on tape
  - scan across tape, crossing of single 0 and single 1
- If 0s still remain after all the 1s have been crossed out, or vice-versa, *reject*. Otherwise, if the tape is empty, *accept*.

Question

So how much time does M_1 need?
Number of steps may depend on several parameters.
Example: if input is a graph, could depend on number of

- nodes
- edges
- maximum degree
- all, some, or none of the above!

Question

Our Gordian knot solution:

Definition: Complexity is measured as function of input string length.

- worst case: longest running time on input of given length
- average case: average running time on given length

Actually, here we consider worst case.

Definition

Let M be a deterministic TM that halts on all inputs. The running time of M is a function

$$f : \mathcal{N} \longrightarrow \mathcal{N}$$

where $f(n)$ is the maximum running time of M on input of length n. Terminology

- M runs in time $f(n)$
- M is an $f(n)$-time TM

Running Time

The exact running time of most algorithms is quite complex.
Better to “estimate” it.
Informally, we want to focus on “important” parts only.
Example:

- $6n^3 + 2n^2 + 20n + 45$ has four terms.
- $6n^3$ more import
- n^3 most important

Asymptotic Notation

Consider functions,

$$f, g : \mathcal{N} \longrightarrow \mathcal{R}^{+}$$

We say that

$$f(n) = O(g(n))$$

if there exist positive integers c and n_0 such that

$$f(n) \leq c \cdot g(n)$$

for $n \geq n_0$.

Confused?

Basic idea: ignore constant factor differences.

- $617n^3 + 277n^2 + 720n + 7n = O(n^3)$.
- $2 = O(1)$
- $\sin(x) = O(1)$.

Reality Check

Consider

$$f_1(n) = 5n^3 + 2n + 22n + 6$$

We claim that

$$f_1(n) = O(n^3)$$

Let $c = 6$ and $n_0 = 10$. Then

$$5n^3 + 2n + 22n + 6 \leq 6n^3$$

for every $n \geq 10$.

Reality Check (Part Two)

Recall:

$$f_1(n) = 5n^3 + 2n + 22n + 6$$

- we have seen that $f_1(n) = O(n^3)$.
- also that $f_1(n) = O(n^4)$.
- but $f_1(n)$ is not $O(n^2)$, because no value for c or n_0 works!

Logarithms

The big-O interacts with logarithms in a particular way.
High-school identity:

$$\log_b n = \frac{\log_2 n}{\log_2 b}$$

- changing b changes only constant factor
- When we say $f(n) = O(\log n)$, the base is unimportant

Important Notation

Sometimes we will see

$$f(n) = O(n^2) + O(n).$$

Each occurrence of O symbol is distinct constant.

But $O(n^2)$ term dominates $O(n)$ term, equivalent to $f(n) = O(n^2)$.

Important Notation

Exponents are even more fun.
What does this mean?

$$f(n) = 2^{O(n)}$$

It means an upper bound of 2^{cn} for some constant c

Important Notion

What does this mean?

$$f(n) = 2^{O(\log n)}$$

Identities

$$\begin{aligned} n &= 2^{\log_2 n} \\ n^c &= 2^{c \log_2 n} \end{aligned}$$

it follows that $2^{O(\log n)}$ means an upper bound of n^c for some constant c

Related Important Notions

- A bound of n^c, where $c > 0$, is called polynomial.
- A bound of $2^{(n^\delta)}$, where $\delta > 0$, is called exponential.