Relations between Complexity Classes

K. Subramani[1]

[1]Lane Department of Computer Science and Electrical Engineering
West Virginia University

General Techniques

Outline

1. Complexity of Classes
   - Specification
   - Complements of complexity classes
2. The Hierarchy Theorem
   - Setup Lemmata
   - The Theorem
   - Consequences of the Hierarchy Theorem

Outline

1. Complexity of Classes
   - Specification
   - Complements of complexity classes
2. The Hierarchy Theorem
   - Setup Lemmata
   - The Theorem
   - Consequences of the Hierarchy Theorem

Outline

1. Complexity of Classes
   - Specification
   - Complements of complexity classes
2. The Hierarchy Theorem
   - Setup Lemmata
   - The Theorem
   - Consequences of the Hierarchy Theorem

Specification

Class Characteristics

(i) Model of Computatiion - multi-string Turing Machine
(ii) Mode of Computation - Deterministic or Non-deterministic
(iii) Resource of interest - Time, space, etc.
(iv) Bound - A function $f : \mathcal{N} \rightarrow \mathcal{N}$.

Definition

A complexity class is the set of all languages decided by a multi-string string Turing machine M operating in the appropriate mode and such that for any input x, M spends at most $f(|x|)$ of the specified resource.

Specification

Class Characteristics

(i) Model of Computatiion - multi-string Turing Machine

(ii) Mode of Computation - Deterministic or Non-deterministic

(iii) Resource of interest - Time, space, etc.

(iv) Bound - A function $f : \mathcal{N} \rightarrow \mathcal{N}$.

Definition

A complexity class is the set of all languages decided by a multi-string string Turing machine M operating in the appropriate mode and such that for any input x, M spends at most $f(|x|)$ of the specified resource.

Specification

Class Characteristics

(i) Model of Computatiion - multi-string Turing Machine
(ii) Mode of Computation - Deterministic or Non-deterministic
(iii) Resource of interest - Time, space, etc.
(iv) Bound - A function $f : \mathcal{N} \rightarrow \mathcal{N}$.

Definition

A complexity class is the set of all languages decided by a multi-string string Turing machine M operating in the appropriate mode and such that for any input x, M spends at most $f(|x|)$ of the specified resource.

Specification

Class Characteristics

(i) Model of Computatiion - multi-string Turing Machine
(ii) Mode of Computation - Deterministic or Non-deterministic
(iii) Resource of interest - Time, space, etc.
(iv) Bound - A function $f : \mathcal{N} \rightarrow \mathcal{N}$.

Definition

A complexity class is the set of all languages decided by a multi-string string Turing machine M operating in the appropriate mode and such that for any input x, M spends at most $f(|x|)$ of the specified resource.

Specification

Class Characteristics

(i) Model of Computatiion - multi-string Turing Machine

(ii) Mode of Computation - Deterministic or Non-deterministic

(iii) Resource of interest - Time, space, etc.

(iv) Bound - A function $f : \mathcal{N} \rightarrow \mathcal{N}$.

Definition

A complexity class is the set of all languages decided by a multi-string string Turing machine M operating in the appropriate mode and such that for any input x, M spends at most $f(|x|)$ of the specified resource.

Specification

Class Characteristics

(i) Model of Computatiion - multi-string Turing Machine

(ii) Mode of Computation - Deterministic or Non-deterministic

(iii) Resource of interest - Time, space, etc.

(iv) Bound - A function $f : \mathcal{N} \rightarrow \mathcal{N}$.

Definition

A complexity class is the set of all languages decided by a multi-string string Turing machine M operating in the appropriate mode and such that for any input x, M spends at most $f(|x|)$ of the specified resource.

Specification

Class Characteristics

(i) Model of Computatiion - multi-string Turing Machine
(ii) Mode of Computation - Deterministic or Non-deterministic
(iii) Resource of interest - Time, space, etc.
(iv) Bound - A function $f : \mathcal{N} \rightarrow \mathcal{N}$.

Definition

A complexity class is the set of all languages decided by a multi-string string Turing machine M operating in the appropriate mode and such that for any input x, M spends at most $f(|x|)$ of the specified resource.

Specification

Class Characteristics

(i) Model of Computatiion - multi-string Turing Machine
(ii) Mode of Computation - Deterministic or Non-deterministic
(iii) Resource of interest - Time, space, etc.
(iv) Bound - A function $f : \mathcal{N} \rightarrow \mathcal{N}$.

Definition

A complexity class is the set of all languages decided by a multi-string string Turing machine M operating in the appropriate mode and such that for any input x, M spends at most $f(|x|)$ of the specified resource.

Specification

Class Characteristics

(i) Model of Computatiion - multi-string Turing Machine
(ii) Mode of Computation - Deterministic or Non-deterministic
(iii) Resource of interest - Time, space, etc.
(iv) Bound - A function $f : \mathcal{N} \rightarrow \mathcal{N}$.

Definition

A complexity class is the set of all languages decided by a multi-string string Turing machine M operating in the appropriate mode and such that for any input x, M spends at most $f(|x|)$ of the specified resource.

Specification

Class Characteristics

(i) Model of Computatiion - multi-string Turing Machine
(ii) Mode of Computation - Deterministic or Non-deterministic
(iii) Resource of interest - Time, space, etc.
(iv) Bound - A function $f : \mathcal{N} \rightarrow \mathcal{N}$.

Definition

A complexity class is the set of all languages decided by a multi-string string Turing machine M operating in the appropriate mode and such that for any input x, M spends at most $f(|x|)$ of the specified resource.

Specification (contd.)

Definition

A function f is said to be a *proper complexity function*, if $(\forall n \geq 0)(f(n+1) \geq f(n))$ and there exists a k-string Turing machine $M_f = (K, \Sigma, \delta, s)$ with input and output, which on input x, computes $\sqcap^{f(|x|)}$ in $O(|x| + f(|x|))$ steps and uses $O(f(|x|))$ space besides its input.

Typical proper complexity functions

c, n, $n!$, $\sqrt{n}$, $\log n$, $\ldots$

Notational Convenience

TIME(f), **SPACE**(f), **NTIME**(f), **NSPACE**(f). Parameterized complexity classes:

(i) $\mathbf{P} = \mathbf{TIME}(n^k) = \cup_{j>0}\mathbf{TIME}(n^j)$.

(ii) $\mathbf{NP} = \mathbf{NTIME}(n^k) = \cup_{j>0}\mathbf{NTIME}(n^j)$.

(iii) $\mathbf{PSPACE} = \mathbf{SPACE}(n^k) = \cup_{j>0}\mathbf{SPACE}(n^j)$.

(iv) $\mathbf{NPSPACE} = \mathbf{NSPACE}(n^k) = \cup_{j>0}\mathbf{NSPACE}(n^j)$.

(v) $\mathbf{EXP} = \mathbf{TIME}(2^{n^k}) = \cup_{j>0}\mathbf{TIME}(2^{n^j})$.

(vi) $\mathbf{L} = \mathbf{SPACE}(\log n)$.

(vii) $\mathbf{NL} = \mathbf{NSPACE}(\log n)$.

Specification (contd.)

Definition

A function f is said to be a *proper complexity function*, if $(\forall n \geq 0)(f(n+1) \geq f(n))$ and there exists a k-string Turing machine $M_f = (K, \Sigma, \delta, s)$ with input and output, which on input x, computes $\sqcap^{f(|x|)}$ in $O(|x| + f(|x|))$ steps and uses $O(f(|x|))$ space besides its input.

Typical proper complexity functions

$c, n, n!, \sqrt{n}, \log n, \ldots$

Notational Convenience

TIME(f), **SPACE**(f), **NTIME**(f), **NSPACE**(f). Parameterized complexity classes:

(i) $\mathbf{P} = \mathbf{TIME}(n^k) = \cup_{j>0}\mathbf{TIME}(n^j)$.

(ii) $\mathbf{NP} = \mathbf{NTIME}(n^k) = \cup_{j>0}\mathbf{NTIME}(n^j)$.

(iii) $\mathbf{PSPACE} = \mathbf{SPACE}(n^k) = \cup_{j>0}\mathbf{SPACE}(n^j)$.

(iv) $\mathbf{NPSPACE} = \mathbf{NSPACE}(n^k) = \cup_{j>0}\mathbf{NSPACE}(n^j)$.

(v) $\mathbf{EXP} = \mathbf{TIME}(2^{n^k}) = \cup_{j>0}\mathbf{TIME}(2^{n^j})$.

(vi) $\mathbf{L} = \mathbf{SPACE}(\log n)$.

(vii) $\mathbf{NL} = \mathbf{NSPACE}(\log n)$.

Specification (contd.)

Definition

A function f is said to be a *proper complexity function*, if $(\forall n \geq 0)(f(n+1) \geq f(n))$ and there exists a k-string Turing machine $M_f = (K, \Sigma, \delta, s)$ with input and output, which on input x, computes $\sqcap^{f(|x|)}$ in $O(|x| + f(|x|))$ steps and uses $O(f(|x|))$ space besides its input.

Typical proper complexity functions

$c, n, n!, \sqrt{n}, \log n, \ldots$

Notational Convenience

TIME(f), **SPACE**(f), **NTIME**(f), **NSPACE**(f). Parameterized complexity classes:

(i) $\mathbf{P} = \mathbf{TIME}(n^k) = \cup_{j>0}\mathbf{TIME}(n^j)$.

(ii) $\mathbf{NP} = \mathbf{NTIME}(n^k) = \cup_{j>0}\mathbf{NTIME}(n^j)$.

(iii) $\mathbf{PSPACE} = \mathbf{SPACE}(n^k) = \cup_{j>0}\mathbf{SPACE}(n^j)$.

(iv) $\mathbf{NPSPACE} = \mathbf{NSPACE}(n^k) = \cup_{j>0}\mathbf{NSPACE}(n^j)$.

(v) $\mathbf{EXP} = \mathbf{TIME}(2^{n^k}) = \cup_{j>0}\mathbf{TIME}(2^{n^j})$.

(vi) $\mathbf{L} = \mathbf{SPACE}(\log n)$.

(vii) $\mathbf{NL} = \mathbf{NSPACE}(\log n)$.

Specification (contd.)

Definition

A function f is said to be a *proper complexity function*, if $(\forall n \geq 0)(f(n+1) \geq f(n))$ and there exists a k-string Turing machine $M_f = (K, \Sigma, \delta, s)$ with input and output, which on input x, computes $\sqcap^{f(|x|)}$ in $O(|x| + f(|x|)$ steps and uses $O(f(|x|)$ space besides its input.

Typical proper complexity functions

c, n, $n!$, $\sqrt{n}$, $\log n$, ...

Notational Convenience

TIME(f), **SPACE**(f), **NTIME**(f), **NSPACE**(f). Parameterized complexity classes:

(i) **P** $=$ **TIME**$(n^k) = \cup_{j>0}$**TIME**(n^j).

(ii) **NP** $=$ **NTIME**$(n^k) = \cup_{j>0}$**NTIME**(n^j).

(iii) **PSPACE** $=$ **SPACE**$(n^k) = \cup_{j>0}$**SPACE**(n^j).

(iv) **NPSPACE** $=$ **NSPACE**$(n^k) = \cup_{j>0}$**NSPACE**(n^j).

(v) **EXP** $=$ **TIME**$(2^{n^k}) = \cup_{j>0}$**TIME**(2^{n^j}).

(vi) **L** $=$ **SPACE**$(\log n)$.

(vii) **NL** $=$ **NSPACE**$(\log n)$.

Specification (contd.)

Definition

A function f is said to be a *proper complexity function*, if $(\forall n \geq 0)(f(n+1) \geq f(n))$ and there exists a k-string Turing machine $M_f = (K, \Sigma, \delta, s)$ with input and output, which on input x, computes $\sqcap^{f(|x|)}$ in $O(|x| + f(|x|)$ steps and uses $O(f(|x|)$ space besides its input.

Typical proper complexity functions

$c, n, n!, \sqrt{n}, \log n, \ldots$

Notational Convenience

TIME(f), **SPACE**(f), **NTIME**(f), **NSPACE**(f). Parameterized complexity classes:

(i) $\mathbf{P} = \mathbf{TIME}(n^k) = \cup_{j>0}\mathbf{TIME}(n^j)$.

(ii) $\mathbf{NP} = \mathbf{NTIME}(n^k) = \cup_{j>0}\mathbf{NTIME}(n^j)$.

(iii) $\mathbf{PSPACE} = \mathbf{SPACE}(n^k) = \cup_{j>0}\mathbf{SPACE}(n^j)$.

(iv) $\mathbf{NPSPACE} = \mathbf{NSPACE}(n^k) = \cup_{j>0}\mathbf{NSPACE}(n^j)$.

(v) $\mathbf{EXP} = \mathbf{TIME}(2^{n^k}) = \cup_{j>0}\mathbf{TIME}(2^{n^j})$.

(vi) $\mathbf{L} = \mathbf{SPACE}(\log n)$.

(vii) $\mathbf{NL} = \mathbf{NSPACE}(\log n)$.

Specification (contd.)

Definition

A function f is said to be a *proper complexity function*, if $(\forall n \geq 0)(f(n+1) \geq f(n))$ and there exists a k-string Turing machine $M_f = (K, \Sigma, \delta, s)$ with input and output, which on input x, computes $\sqcap^{f(|x|)}$ in $O(|x| + f(|x|))$ steps and uses $O(f(|x|))$ space besides its input.

Typical proper complexity functions

c, n, $n!$, $\sqrt{n}$, $\log n$, $\ldots$

Notational Convenience

TIME(f), **SPACE**(f), **NTIME**(f), **NSPACE**(f). Parameterized complexity classes:

(i) $\mathbf{P} = \mathbf{TIME}(n^k) = \cup_{j>0}\mathbf{TIME}(n^j)$.

(ii) $\mathbf{NP} = \mathbf{NTIME}(n^k) = \cup_{j>0}\mathbf{NTIME}(n^j)$.

(iii) $\mathbf{PSPACE} = \mathbf{SPACE}(n^k) = \cup_{j>0}\mathbf{SPACE}(n^j)$.

(iv) $\mathbf{NPSPACE} = \mathbf{NSPACE}(n^k) = \cup_{j>0}\mathbf{NSPACE}(n^j)$.

(v) $\mathbf{EXP} = \mathbf{TIME}(2^{n^k}) = \cup_{j>0}\mathbf{TIME}(2^{n^j})$.

(vi) $\mathbf{L} = \mathbf{SPACE}(\log n)$.

(vii) $\mathbf{NL} = \mathbf{NSPACE}(\log n)$.

Specification (contd.)

Definition

A function f is said to be a *proper complexity function*, if $(\forall n \geq 0)(f(n+1) \geq f(n))$ and there exists a k-string Turing machine $M_f = (K, \Sigma, \delta, s)$ with input and output, which on input x, computes $\sqcap^{f(|x|)}$ in $O(|x| + f(|x|))$ steps and uses $O(f(|x|))$ space besides its input.

Typical proper complexity functions

c, n, $n!$, $\sqrt{n}$, $\log n$, ...

Notational Convenience

TIME(f), **SPACE**(f), **NTIME**(f), **NSPACE**(f). Parameterized complexity classes:

(i) $\mathbf{P} = \mathbf{TIME}(n^k) = \cup_{j>0}\mathbf{TIME}(n^j)$.

(ii) $\mathbf{NP} = \mathbf{NTIME}(n^k) = \cup_{j>0}\mathbf{NTIME}(n^j)$.

(iii) $\mathbf{PSPACE} = \mathbf{SPACE}(n^k) = \cup_{j>0}\mathbf{SPACE}(n^j)$.

(iv) $\mathbf{NPSPACE} = \mathbf{NSPACE}(n^k) = \cup_{j>0}\mathbf{NSPACE}(n^j)$.

(v) $\mathbf{EXP} = \mathbf{TIME}(2^{n^k}) = \cup_{j>0}\mathbf{TIME}(2^{n^j})$.

(vi) $\mathbf{L} = \mathbf{SPACE}(\log n)$.

(vii) $\mathbf{NL} = \mathbf{NSPACE}(\log n)$.

Specification (contd.)

Definition

A function f is said to be a *proper complexity function*, if $(\forall n \geq 0)(f(n+1) \geq f(n))$ and there exists a k-string Turing machine $M_f = (K, \Sigma, \delta, s)$ with input and output, which on input x, computes $\sqcap^{f(|x|)}$ in $O(|x| + f(|x|)$ steps and uses $O(f(|x|)$ space besides its input.

Typical proper complexity functions

c, n, $n!$, $\sqrt{n}$, $\log n$, $\ldots$

Notational Convenience

TIME(f), **SPACE**(f), **NTIME**(f), **NSPACE**(f). Parameterized complexity classes:

(i) $\mathbf{P} = \mathbf{TIME}(n^k) = \cup_{j>0}\mathbf{TIME}(n^j)$.

(ii) $\mathbf{NP} = \mathbf{NTIME}(n^k) = \cup_{j>0}\mathbf{NTIME}(n^j)$.

(iii) $\mathbf{PSPACE} = \mathbf{SPACE}(n^k) = \cup_{j>0}\mathbf{SPACE}(n^j)$.

(iv) $\mathbf{NPSPACE} = \mathbf{NSPACE}(n^k) = \cup_{j>0}\mathbf{NSPACE}(n^j)$.

(v) $\mathbf{EXP} = \mathbf{TIME}(2^{n^k}) = \cup_{j>0}\mathbf{TIME}(2^{n^j})$.

(vi) $\mathbf{L} = \mathbf{SPACE}(\log n)$.

(vii) $\mathbf{NL} = \mathbf{NSPACE}(\log n)$.

Specification (contd.)

Definition

A function f is said to be a *proper complexity function*, if $(\forall n \geq 0)(f(n+1) \geq f(n))$ and there exists a k-string Turing machine $M_f = (K, \Sigma, \delta, s)$ with input and output, which on input x, computes $\sqcap^{f(|x|)}$ in $O(|x| + f(|x|))$ steps and uses $O(f(|x|))$ space besides its input.

Typical proper complexity functions

$c, n, n!, \sqrt{n}, \log n, \ldots$

Notational Convenience

TIME(f), **SPACE**(f), **NTIME**(f), **NSPACE**(f). Parameterized complexity classes:

(i) $\mathbf{P} = \mathbf{TIME}(n^k) = \cup_{j>0}\mathbf{TIME}(n^j)$.

(ii) $\mathbf{NP} = \mathbf{NTIME}(n^k) = \cup_{j>0}\mathbf{NTIME}(n^j)$.

(iii) $\mathbf{PSPACE} = \mathbf{SPACE}(n^k) = \cup_{j>0}\mathbf{SPACE}(n^j)$.

(iv) $\mathbf{NPSPACE} = \mathbf{NSPACE}(n^k) = \cup_{j>0}\mathbf{NSPACE}(n^j)$.

(v) $\mathbf{EXP} = \mathbf{TIME}(2^{n^k}) = \cup_{j>0}\mathbf{TIME}(2^{n^j})$.

(vi) $\mathbf{L} = \mathbf{SPACE}(\log n)$.

(vii) $\mathbf{NL} = \mathbf{NSPACE}(\log n)$.

Specification (contd.)

Definition

A function f is said to be a *proper complexity function*, if $(\forall n \geq 0)(f(n+1) \geq f(n))$ and there exists a k-string Turing machine $M_f = (K, \Sigma, \delta, s)$ with input and output, which on input x, computes $\sqcap^{f(|x|)}$ in $O(|x| + f(|x|)$ steps and uses $O(f(|x|)$ space besides its input.

Typical proper complexity functions

$c, n, n!, \sqrt{n}, \log n, \ldots$

Notational Convenience

TIME(f), **SPACE**(f), **NTIME**(f), **NSPACE**(f). Parameterized complexity classes:

(i) $\mathbf{P} = \mathbf{TIME}(n^k) = \cup_{j>0}\mathbf{TIME}(n^j)$.

(ii) $\mathbf{NP} = \mathbf{NTIME}(n^k) = \cup_{j>0}\mathbf{NTIME}(n^j)$.

(iii) $\mathbf{PSPACE} = \mathbf{SPACE}(n^k) = \cup_{j>0}\mathbf{SPACE}(n^j)$.

(iv) $\mathbf{NPSPACE} = \mathbf{NSPACE}(n^k) = \cup_{j>0}\mathbf{NSPACE}(n^j)$.

(v) $\mathbf{EXP} = \mathbf{TIME}(2^{n^k}) = \cup_{j>0}\mathbf{TIME}(2^{n^j})$.

(vi) $\mathbf{L} = \mathbf{SPACE}(\log n)$.

(vii) $\mathbf{NL} = \mathbf{NSPACE}(\log n)$.

Specification (contd.)

Definition

A function f is said to be a *proper complexity function*, if $(\forall n \geq 0)(f(n+1) \geq f(n))$ and there exists a k-string Turing machine $M_f = (K, \Sigma, \delta, s)$ with input and output, which on input x, computes $\sqcap^{f(|x|)}$ in $O(|x| + f(|x|)$ steps and uses $O(f(|x|)$ space besides its input.

Typical proper complexity functions

$c, n, n!, \sqrt{n}, \log n, \ldots$

Notational Convenience

TIME(f), **SPACE**(f), **NTIME**(f), **NSPACE**(f). Parameterized complexity classes:

(i) $\mathbf{P} = \mathbf{TIME}(n^k) = \cup_{j>0}\mathbf{TIME}(n^j)$.

(ii) $\mathbf{NP} = \mathbf{NTIME}(n^k) = \cup_{j>0}\mathbf{NTIME}(n^j)$.

(iii) $\mathbf{PSPACE} = \mathbf{SPACE}(n^k) = \cup_{j>0}\mathbf{SPACE}(n^j)$.

(iv) $\mathbf{NPSPACE} = \mathbf{NSPACE}(n^k) = \cup_{j>0}\mathbf{NSPACE}(n^j)$.

(v) $\mathbf{EXP} = \mathbf{TIME}(2^{n^k}) = \cup_{j>0}\mathbf{TIME}(2^{n^j})$.

(vi) $\mathbf{L} = \mathbf{SPACE}(\log n)$.

(vii) $\mathbf{NL} = \mathbf{NSPACE}(\log n)$.

Outline

1. Complexity of Classes
   - Specification
   - Complements of complexity classes
2. The Hierarchy Theorem
   - Setup Lemmata
   - The Theorem
   - Consequences of the Hierarchy Theorem

Complementing Languages, Problems and Complexity Classes

Definition

Let $L \subseteq \Sigma^*$ denote a language. The complement of L denoted by $\bar{L} = \Sigma^* - L$.

Definition

The complement of a decision problem A, called A-COMPLEMENT, is the problem whose "yes"-instances and "no"-instances correspond to the "no"-instances and "yes"-instances of A respectively, e.g. SAT COMPLEMENT, Hamilton Path COMPLEMENT, ...

Definition

The complement of a complexity class $\mathcal{C}$, called **co**$\mathcal{C}$ denotes the class $\{\bar{L} : L \in \mathcal{C}\}$.

Complementing Languages, Problems and Complexity Classes

Definition

Let $L \subseteq \Sigma^*$ denote a language. The complement of L denoted by $\bar{L} = \Sigma^* - L$.

Definition

The complement of a decision problem A, called A-COMPLEMENT, is the problem whose "yes"-instances and "no"-instances correspond to the "no"-instances and "yes"-instances of A respectively, e.g. SAT COMPLEMENT, Hamilton Path COMPLEMENT, ...

Definition

The complement of a complexity class $\mathcal{C}$, called **co**$\mathcal{C}$ denotes the class $\{\bar{L} : L \in \mathcal{C}\}$.

Complementing Languages, Problems and Complexity Classes

Definition

Let $L \subseteq \Sigma^*$ denote a language. The complement of L denoted by $\bar{L} = \Sigma^* - L$.

Definition

The complement of a decision problem A, called A-COMPLEMENT, is the problem whose "yes"-instances and "no"-instances correspond to the "no"-instances and "yes"-instances of A respectively, e.g. SAT COMPLEMENT, Hamilton Path COMPLEMENT, ...

Definition

The complement of a complexity class $\mathcal{C}$, called **co**$\mathcal{C}$ denotes the class $\{\bar{L} : L \in \mathcal{C}\}$.

Complement of Complexity Classes

Relationship between $\mathcal{C}$ and **co**$\mathcal{C}$

(i) $\mathcal{C}$ is deterministic.

(ii) $\mathcal{C}$ is non-deterministic.

Complement of Complexity Classes

Relationship between $\mathcal{C}$ and **co**$\mathcal{C}$

(i) $\mathcal{C}$ is deterministic.

(ii) $\mathcal{C}$ is non-deterministic.

Complement of Complexity Classes

Relationship between $\mathcal{C}$ and $\mathbf{co}\mathcal{C}$

(i) $\mathcal{C}$ is deterministic.

(ii) $\mathcal{C}$ is non-deterministic.

Outline

1. Complexity of Classes
   - Specification
   - Complements of complexity classes
2. The Hierarchy Theorem
   - Setup Lemmata
   - The Theorem
   - Consequences of the Hierarchy Theorem

Setup Lemmata

Goal

To show that with sufficiently greater time, Turing machines can in fact perform more complex computational tasks.

Setup

Let $f(n) \geq n$ denote a property complexity function. We define the the f-bounded Halting Problem, H_f as follows: $H_f = \{\langle M; x\rangle : M$ accepts input x after are most $f(|x|)$ steps $\}$.

Setup Lemmata

Goal

To show that with sufficiently greater time, Turing machines can in fact perform more complex computational tasks.

Setup

Let $f(n) \geq n$ denote a property complexity function. We define the the f-bounded Halting Problem, H_f as follows: $H_f = \{\langle M; x\rangle : M$ accepts input x after are most $f(|x|)$ steps $\}$.

Setup Lemmata

Goal

To show that with sufficiently greater time, Turing machines can in fact perform more complex computational tasks.

Setup

Let $f(n) \geq n$ denote a property complexity function. We define the the f-bounded Halting Problem, H_f as follows: $H_f = \{\langle M; x\rangle : M$ accepts input x after are most $f(|x|)$ steps $\}$.

Setup Lemmata (contd.)

Lemma

$H_f \in \textbf{TIME}((f(n))^3)$.

Proof.

Use a 4-string Turing Machine U_f that is a combination of the following machines:
(a) The Universal Turing machine, (b) The single-string simulator of multi-string Turing machines, (c) The linear speedup machine, (d) The "yardstick" machine that computes $f(n)$ precisely.
Approach:

(i) U_f copies x onto its first string and then uses M_f to initialize its 4^{th} string with $\sqcap^{f(|x|)}$. (Each move of M is marked off on this string.) Total time used thus far is $O(|x| + f(|x|))$.
(ii) U_f also copies M on its third string and s on its second string.
(iii) U_f then simulates M on x, precisely as the Universal Turing Machine does.
(iv) A single move of M takes $O(l_M \cdot {k_M}^2 \cdot f(|x|) = O((f(n))^2)$ steps, where l_M is the length of the description of each state and symbol of M and k_M is the number of strings of M.
(v) Entire simulation takes $O((f(n))^3)$ time which can be made precisely $(f(n))^3$ using linear speedup.

□

Setup Lemmata (contd.)

Lemma

$H_f \in \mathbf{TIME}((f(n))^3)$.

Proof.

Use a 4-string Turing Machine U_f that is a combination of the following machines:
(a) The Universal Turing machine, (b) The single-string simulator of multi-string Turing machines, (c) The linear speedup machine, (d) The "yardstick" machine that computes $f(n)$ precisely.
Approach:

(i) U_f copies x onto its first string and then uses M_f to initialize its 4^{th} string with $\sqcap^{f(|x|)}$. (Each move of M is marked off on this string.) Total time used thus far is $O(|x| + f(|x|))$.
(ii) U_f also copies M on its third string and s on its second string.
(iii) U_f then simulates M on x, precisely as the Universal Turing Machine does.
(iv) A single move of M takes $O(l_M \cdot k_M{}^2 \cdot f(|x|) = O((f(n))^2)$ steps, where l_M is the length of the description of each state and symbol of M and k_M is the number of strings of M.
(v) Entire simulation takes $O((f(n))^3)$ time which can be made precisely $(f(n))^3$ using linear speedup.

Setup Lemmata (contd.)

Lemma

$H_f \in \textbf{TIME}((f(n))^3)$.

Proof.

Use a 4-string Turing Machine U_f that is a combination of the following machines:
(a) The Universal Turing machine, (b) The single-string simulator of multi-string Turing machines,
(c) The linear speedup machine, (d) The "yardstick" machine that computes $f(n)$ precisely.
Approach:

(i) U_f copies x onto its first string and then uses M_f to initialize its 4^{th} string with $\sqcap^{f(|x|)}$. (Each move of M is marked off on this string.) Total time used thus far is $O(|x| + f(|x|))$.

(ii) U_f also copies M on its third string and s on its second string.

(iii) U_f then simulates M on x, precisely as the Universal Turing Machine does.

(iv) A single move of M takes $O(l_M \cdot k_M^2 \cdot f(|x|) = O((f(n))^2)$ steps, where l_M is the length of the description of each state and symbol of M and k_M is the number of strings of M.

(v) Entire simulation takes $O((f(n))^3)$ time which can be made precisely $(f(n))^3$ using linear speedup.

Setup Lemmata (contd.)

Lemma

$H_f \in \mathbf{TIME}((f(n))^3)$.

Proof.

Use a 4-string Turing Machine U_f that is a combination of the following machines:
(a) The Universal Turing machine, (b) The single-string simulator of multi-string Turing machines,
(c) The linear speedup machine, (d) The "yardstick" machine that computes $f(n)$ precisely.
Approach:

(i) U_f copies x onto its first string and then uses M_f to initialize its 4^{th} string with $\sqcap^{f(|x|)}$. (Each move of M is marked off on this string.) Total time used thus far is $O(|x| + f(|x|))$.

(ii) U_f also copies M on its third string and s on its second string.

(iii) U_f then simulates M on x, precisely as the Universal Turing Machine does.

(iv) A single move of M takes $O(l_M \cdot k_M{}^2 \cdot f(|x|) = O((f(n))^2)$ steps, where l_M is the length of the description of each state and symbol of M and k_M is the number of strings of M.

(v) Entire simulation takes $O((f(n))^3)$ time which can be made precisely $(f(n))^3$ using linear speedup.

Setup Lemmata (contd.)

Lemma

$H_f \in$ **TIME**$((f(n))^3)$.

Proof.

Use a 4-string Turing Machine U_f that is a combination of the following machines:
(a) The Universal Turing machine, (b) The single-string simulator of multi-string Turing machines, (c) The linear speedup machine, (d) The "yardstick" machine that computes $f(n)$ precisely.
Approach:

(i) U_f copies x onto its first string and then uses M_f to initialize its 4^{th} string with $\sqcap^{f(|x|)}$. (Each move of M is marked off on this string.) Total time used thus far is $O(|x| + f(|x|))$.

(ii) U_f also copies M on its third string and s on its second string.

(iii) U_f then simulates M on x, precisely as the Universal Turing Machine does.

(iv) A single move of M takes $O(l_M \cdot k_M{}^2 \cdot f(|x|) = O((f(n))^2)$ steps, where l_M is the length of the description of each state and symbol of M and k_M is the number of strings of M.

(v) Entire simulation takes $O((f(n))^3)$ time which can be made precisely $(f(n))^3$ using linear speedup.

Setup Lemmata (contd.)

Lemma

$H_f \in \textbf{TIME}((f(n))^3)$.

Proof.

Use a 4-string Turing Machine U_f that is a combination of the following machines:
(a) The Universal Turing machine, (b) The single-string simulator of multi-string Turing machines, (c) The linear speedup machine, (d) The "yardstick" machine that computes $f(n)$ precisely.
Approach:

(i) U_f copies x onto its first string and then uses M_f to initialize its 4^{th} string with $\sqcap^{f(|x|)}$. (Each move of M is marked off on this string.) Total time used thus far is $O(|x| + f(|x|))$.

(ii) U_f also copies M on its third string and s on its second string.

(iii) U_f then simulates M on x, precisely as the Universal Turing Machine does.

(iv) A single move of M takes $O(l_M \cdot {k_M}^2 \cdot f(|x|) = O((f(n))^2)$ steps, where l_M is the length of the description of each state and symbol of M and k_M is the number of strings of M.

(v) Entire simulation takes $O((f(n))^3)$ time which can be made precisely $(f(n))^3$ using linear speedup.

Setup Lemmata (contd.)

Lemma

$H_f \in \textbf{TIME}((f(n))^3)$.

Proof.

Use a 4-string Turing Machine U_f that is a combination of the following machines:
(a) The Universal Turing machine, (b) The single-string simulator of multi-string Turing machines, (c) The linear speedup machine, (d) The "yardstick" machine that computes $f(n)$ precisely.
Approach:

(i) U_f copies x onto its first string and then uses M_f to initialize its 4^{th} string with $\sqcap^{f(|x|)}$. (Each move of M is marked off on this string.) Total time used thus far is $O(|x| + f(|x|))$.

(ii) U_f also copies M on its third string and s on its second string.

(iii) U_f then simulates M on x, precisely as the Universal Turing Machine does.

(iv) A single move of M takes $O(l_M \cdot k_M{}^2 \cdot f(|x|) = O((f(n))^2)$ steps, where l_M is the length of the description of each state and symbol of M and k_M is the number of strings of M.

(v) Entire simulation takes $O((f(n))^3)$ time which can be made precisely $(f(n))^3$ using linear speedup.

Setup Lemmata (contd.)

Lemma

$H_f \in \mathbf{TIME}((f(n))^3)$.

Proof.

Use a 4-string Turing Machine U_f that is a combination of the following machines:
(a) The Universal Turing machine, (b) The single-string simulator of multi-string Turing machines, (c) The linear speedup machine, (d) The "yardstick" machine that computes $f(n)$ precisely.
Approach:

(i) U_f copies x onto its first string and then uses M_f to initialize its 4^{th} string with $\sqcap^{f(|x|)}$. (Each move of M is marked off on this string.) Total time used thus far is $O(|x| + f(|x|))$.

(ii) U_f also copies M on its third string and s on its second string.

(iii) U_f then simulates M on x, precisely as the Universal Turing Machine does.

(iv) A single move of M takes $O(l_M \cdot {k_M}^2 \cdot f(|x|) = O((f(n))^2)$ steps, where l_M is the length of the description of each state and symbol of M and k_M is the number of strings of M.

(v) Entire simulation takes $O((f(n))^3)$ time which can be made precisely $(f(n))^3$ using linear speedup.

Setup Lemmata (contd.)

Lemma

$H_f \in \mathbf{TIME}((f(n))^3)$.

Proof.

Use a 4-string Turing Machine U_f that is a combination of the following machines:
(a) The Universal Turing machine, (b) The single-string simulator of multi-string Turing machines, (c) The linear speedup machine, (d) The "yardstick" machine that computes $f(n)$ precisely.
Approach:

(i) U_f copies x onto its first string and then uses M_f to initialize its 4^{th} string with $\sqcap^{f(|x|)}$. (Each move of M is marked off on this string.) Total time used thus far is $O(|x| + f(|x|))$.

(ii) U_f also copies M on its third string and s on its second string.

(iii) U_f then simulates M on x, precisely as the Universal Turing Machine does.

(iv) A single move of M takes $O(l_M \cdot {k_M}^2 \cdot f(|x|) = O((f(n))^2)$ steps, where l_M is the length of the description of each state and symbol of M and k_M is the number of strings of M.

(v) Entire simulation takes $O((f(n))^3)$ time which can be made precisely $(f(n))^3$ using linear speedup.

Setup Lemmata (contd.)

Lemma

$H_f \in \textbf{TIME}((f(n))^3)$.

Proof.

Use a 4-string Turing Machine U_f that is a combination of the following machines:
(a) The Universal Turing machine, (b) The single-string simulator of multi-string Turing machines, (c) The linear speedup machine, (d) The "yardstick" machine that computes $f(n)$ precisely.
Approach:

(i) U_f copies x onto its first string and then uses M_f to initialize its 4^{th} string with $\sqcap^{f(|x|)}$. (Each move of M is marked off on this string.) Total time used thus far is $O(|x| + f(|x|))$.

(ii) U_f also copies M on its third string and s on its second string.

(iii) U_f then simulates M on x, precisely as the Universal Turing Machine does.

(iv) A single move of M takes $O(l_M \cdot {k_M}^2 \cdot f(|x|) = O((f(n))^2)$ steps, where l_M is the length of the description of each state and symbol of M and k_M is the number of strings of M.

(v) Entire simulation takes $O((f(n))^3)$ time which can be made precisely $(f(n))^3$ using linear speedup.

□

Setup Lemmata (contd.)

Lemma

$H_f \in \mathbf{TIME}((f(n))^3)$.

Proof.

Use a 4-string Turing Machine U_f that is a combination of the following machines:
(a) The Universal Turing machine, (b) The single-string simulator of multi-string Turing machines, (c) The linear speedup machine, (d) The "yardstick" machine that computes $f(n)$ precisely.
Approach:

(i) U_f copies x onto its first string and then uses M_f to initialize its 4^{th} string with $\sqcap^{f(|x|)}$. (Each move of M is marked off on this string.) Total time used thus far is $O(|x| + f(|x|))$.

(ii) U_f also copies M on its third string and s on its second string.

(iii) U_f then simulates M on x, precisely as the Universal Turing Machine does.

(iv) A single move of M takes $O(l_M \cdot {k_M}^2 \cdot f(|x|) = O((f(n))^2)$ steps, where l_M is the length of the description of each state and symbol of M and k_M is the number of strings of M.

(v) Entire simulation takes $O((f(n))^3)$ time which can be made precisely $(f(n))^3$ using linear speedup.

$\square$

Setup Lemmata (contd.)

Lemma

$H_f \notin \textbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Proof.

Assume that there exists a Turing machine M_{H_f} that decides H_f in time $f(\lfloor \frac{n}{2} \rfloor)$. Construct D_f as follows:

$$D_f(M): \ \textbf{if } M_{H_f}(M; M) = \text{“}yes\text{”} \textbf{ then } \text{“}no\text{”} \textbf{ else } \text{“}yes\text{”}.$$

$D_f(M)$ takes the same time as $M_{H_f}(M; M)$ which is $f(\lfloor \frac{2n+1}{2} \rfloor) = f(n)$. What about $D_f(D_f)$? If $(D_f(D_f)) = \text{“}yes\text{”}$, then $M_{H_f}(D_f; D_f) = \text{“}no\text{”}$ and hence $\langle D_f ; D_f \rangle \notin H_f$. But this means that D_f does not accept its description in $f(n)$ steps, i.e., $D_f(D_f) = \text{“}no\text{”}$! Similarly, $D_f(D_f) = \text{“}no\text{”}$ implies $D_f(D_f) = \text{“}yes\text{”}$. It follows that $H_f \notin \textbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$. □

Setup Lemmata (contd.)

Lemma

$H_f \notin \textbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Proof.

Assume that there exists a Turing machine M_{H_f} that decides H_f in time $f(\lfloor \frac{n}{2} \rfloor)$. Construct D_f as follows:

$$D_f(M): \textbf{ if } M_{H_f}(M; M) = \text{“}yes\text{”} \textbf{ then } \text{“}no\text{”} \textbf{ else } \text{“}yes\text{”}.$$

$D_f(M)$ takes the same time as $M_{H_f}(M;\ M)$ which is $f(\lfloor \frac{2n+1}{2} \rfloor) = f(n)$. What about $D_f(D_f)$? If $(D_f(D_f)) = \text{“}yes\text{”}$, then $M_{H_f}(D_f;\ D_f) = \text{“}no\text{”}$ and hence $\langle D_f\ ; D_f \rangle \notin H_f$. But this means that D_f does not accept its description in $f(n)$ steps, i.e., $D_f(D_f) = \text{“}no\text{”}$! Similarly, $D_f(D_f) = \text{“}no\text{”}$ implies $D_f(D_f) = \text{“}yes\text{”}$. It follows that $H_f \notin \textbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$. □

Setup Lemmata (contd.)

Lemma

$H_f \not\in \textbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Proof.

Assume that there exists a Turing machine M_{H_f} that decides H_f in time $f(\lfloor \frac{n}{2} \rfloor)$. Construct D_f as follows:

$$D_f(M) : \textbf{ if } M_{H_f}(M; M) = \text{“}yes\text{”} \textbf{ then } \text{“}no\text{”} \textbf{ else } \text{“}yes\text{”}.$$

$D_f(M)$ takes the same time as $M_{H_f}(M;\ M)$ which is $f(\lfloor \frac{2n+1}{2} \rfloor) = f(n)$. What about $D_f(D_f)$? If $(D_f(D_f)) =$ “yes”, then $M_{H_f}(D_f;\ D_f) =$ “no” and hence $\langle D_f\ ; D_f \rangle \not\in H_f$. But this means that D_f does not accept its description in $f(n)$ steps, i.e., $D_f(D_f) =$ “no”! Similarly, $D_f(D_f) =$ “no” implies $D_f(D_f) =$ “yes”. It follows that $H_f \not\in \textbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$. □

Setup Lemmata (contd.)

Lemma

$H_f \notin \mathbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Proof.

Assume that there exists a Turing machine M_{H_f} that decides H_f in time $f(\lfloor \frac{n}{2} \rfloor)$. Construct D_f as follows:

$$D_f(M): \ \textbf{if } M_{H_f}(M;M) = \text{"}yes\text{"} \textbf{ then } \text{"}no\text{"} \textbf{ else } \text{"}yes\text{"}.$$

$D_f(M)$ takes the same time as $M_{H_f}(M;\ M)$ which is $f(\lfloor \frac{2n+1}{2} \rfloor) = f(n)$. What about $D_f(D_f)$? If $(D_f(D_f)) =$ "*yes*", then $M_{H_f}(D_f;\ D_f) =$ "*no*" and hence $\langle D_f\ ;D_f \rangle \notin H_f$. But this means that D_f does not accept its description in $f(n)$ steps, i.e., $D_f(D_f) =$ "*no*"! Similarly, $D_f(D_f) =$ "*no*" implies $D_f(D_f) =$ "*yes*". It follows that $H_f \notin \mathbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$. □

Setup Lemmata (contd.)

Lemma

$H_f \not\in \mathbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Proof.

Assume that there exists a Turing machine M_{H_f} that decides H_f in time $f(\lfloor \frac{n}{2} \rfloor)$. Construct D_f as follows:

$$D_f(M): \ \textbf{if } M_{H_f}(M; M) = \text{“}yes\text{”} \textbf{ then } \text{“}no\text{”} \textbf{ else } \text{“}yes\text{”}.$$

$D_f(M)$ takes the same time as $M_{H_f}(M;\ M)$ which is $f(\lfloor \frac{2n+1}{2} \rfloor) = f(n)$. What about $D_f(D_f)$? If $(D_f(D_f)) = \text{“}yes\text{”}$, then $M_{H_f}(D_f;\ D_f) = \text{“}no\text{”}$ and hence $\langle D_f ; D_f \rangle \not\in H_f$. But this means that D_f does not accept its description in $f(n)$ steps, i.e., $D_f(D_f) = \text{“}no\text{”}$! Similarly, $D_f(D_f) = \text{“}no\text{”}$ implies $D_f(D_f) = \text{“}yes\text{”}$. It follows that $H_f \not\in \mathbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Setup Lemmata (contd.)

Lemma

$H_f \notin \textbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Proof.

Assume that there exists a Turing machine M_{H_f} that decides H_f in time $f(\lfloor \frac{n}{2} \rfloor)$. Construct D_f as follows:

$$D_f(M): \textbf{ if } M_{H_f}(M;M) = \text{"}yes\text{"} \textbf{ then } \text{"}no\text{"} \textbf{ else } \text{"}yes\text{"}.$$

$D_f(M)$ takes the same time as $M_{H_f}(M;\ M)$ which is $f(\lfloor \frac{2n+1}{2} \rfloor) = f(n)$. What about $D_f(D_f)$? If $(D_f(D_f)) = \text{"}yes\text{"}$, then $M_{H_f}(D_f;\ D_f) = \text{"}no\text{"}$ and hence $\langle D_f\ ; D_f \rangle \notin H_f$. But this means that D_f does not accept its description in $f(n)$ steps, i.e., $D_f(D_f) = \text{"}no\text{"}$! Similarly, $D_f(D_f) = \text{"}no\text{"}$ implies $D_f(D_f) = \text{"}yes\text{"}$. It follows that $H_f \notin \textbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Setup Lemmata (contd.)

Lemma

$H_f \notin \textbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Proof.

Assume that there exists a Turing machine M_{H_f} that decides H_f in time $f(\lfloor \frac{n}{2} \rfloor)$. Construct D_f as follows:

$$D_f(M): \textbf{ if } M_{H_f}(M; M) = \text{"}yes\text{"} \textbf{ then } \text{"}no\text{"} \textbf{ else } \text{"}yes\text{"}.$$

$D_f(M)$ takes the same time as $M_{H_f}(M;\ M)$ which is $f(\lfloor \frac{2n+1}{2} \rfloor) = f(n)$. What about $D_f(D_f)$? If $(D_f(D_f)) = \text{"}yes\text{"}$, then $M_{H_f}(D_f;\ D_f) = \text{"}no\text{"}$ and hence $\langle D_f\ ; D_f \rangle \notin H_f$. But this means that D_f does not accept its description in $f(n)$ steps, i.e., $D_f(D_f) = \text{"}no\text{"}$! Similarly, $D_f(D_f) = \text{"}no\text{"}$ implies $D_f(D_f) = \text{"}yes\text{"}$. It follows that $H_f \notin \textbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$. □

Setup Lemmata (contd.)

Lemma

$H_f \notin \mathbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Proof.

Assume that there exists a Turing machine M_{H_f} that decides H_f in time $f(\lfloor \frac{n}{2} \rfloor)$. Construct D_f as follows:

$$D_f(M): \ \textbf{if } M_{H_f}(M;M) = \text{``}yes\text{''} \textbf{ then } \text{``}no\text{''} \textbf{ else } \text{``}yes\text{''}.$$

$D_f(M)$ takes the same time as $M_{H_f}(M;\ M)$ which is $f(\lfloor \frac{2n+1}{2} \rfloor) = f(n)$. What about $D_f(D_f)$? If $(D_f(D_f)) = \text{``}yes\text{''}$, then $M_{H_f}(D_f;\ D_f) = \text{``}no\text{''}$ and hence $\langle D_f\ ;\ D_f \rangle \notin H_f$. But this means that D_f does not accept its description in $f(n)$ steps, i.e., $D_f(D_f) = \text{``}no\text{''}$! Similarly, $D_f(D_f) = \text{``}no\text{''}$ implies $D_f(D_f) = \text{``}yes\text{''}$. It follows that $H_f \notin \mathbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Setup Lemmata (contd.)

Lemma

$H_f \notin \mathbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$.

Proof.

Assume that there exists a Turing machine M_{H_f} that decides H_f in time $f(\lfloor \frac{n}{2} \rfloor)$. Construct D_f as follows:

$$D_f(M): \ \textbf{if } M_{H_f}(M; M) = \text{“}yes\text{''} \textbf{ then } \text{“}no\text{''} \textbf{ else } \text{“}yes\text{''}.$$

$D_f(M)$ takes the same time as $M_{H_f}(M;\ M)$ which is $f(\lfloor \frac{2n+1}{2} \rfloor) = f(n)$. What about $D_f(D_f)$? If $(D_f(D_f)) = \text{“}yes\text{''}$, then $M_{H_f}(D_f;\ D_f) = \text{“}no\text{''}$ and hence $\langle D_f\ ; D_f \rangle \notin H_f$. But this means that D_f does not accept its description in $f(n)$ steps, i.e., $D_f(D_f) = \text{“}no\text{''}$! Similarly, $D_f(D_f) = \text{“}no\text{''}$ implies $D_f(D_f) = \text{“}yes\text{''}$. It follows that $H_f \notin \mathbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$. $\square$

Outline

1. Complexity of Classes
   - Specification
   - Complements of complexity classes
2. The Hierarchy Theorem
   - Setup Lemmata
   - The Theorem
   - Consequences of the Hierarchy Theorem

The Hierarchy Theorem

Theorem (The Time Hierarchy Theorem)

If $f(n) \geq n$ is a proper complexity function, then the class $\mathbf{TIME}(f(n))$ *is strictly contained in the class* $\mathbf{TIME}((f(2n+1))^3)$.

The Hierarchy Theorem

Theorem (The Time Hierarchy Theorem)

If $f(n) \geq n$ is a proper complexity function, then the class $\mathbf{TIME}(f(n))$ *is strictly contained in the class* $\mathbf{TIME}((f(2n+1))^3)$.

Outline

1. Complexity of Classes
   - Specification
   - Complements of complexity classes
2. The Hierarchy Theorem
   - Setup Lemmata
   - The Theorem
   - Consequences of the Hierarchy Theorem

Consequences of the Hierarchy Theorem

Lemma

P *is a proper subset of* **EXP**.

Proof.

Clearly $\mathbf{P} \subseteq \mathbf{TIME}(2^n)$. As per the Hierarchy theorem, $\mathbf{TIME}(2^n) \subset \mathbf{TIME}((2^{(2n+1)})^3)$. But $\mathbf{TIME}((2^{(2n+1)})^3) \subset \mathbf{TIME}(2^{n^2}) \subseteq \mathbf{EXP}$! □

Lemma

If $f(n)$ *is a proper complexity function, then* $\mathbf{SPACE}(f(n))$ *is proper subset of* $\mathbf{SPACE}(f(n) \cdot \log f(n))$.

Theorem (The Gap Theorem)

There exists a recursive function $f : \mathcal{N} \rightarrow \mathcal{N}$ *such that* $\mathbf{TIME}(f(n)) = \mathbf{TIME}(2^{f(n)})$.

Consequences of the Hierarchy Theorem

Lemma

P *is a proper subset of* **EXP**.

Proof.

Clearly $\mathbf{P} \subseteq \mathbf{TIME}(2^n)$. As per the Hierarchy theorem, $\mathbf{TIME}(2^n) \subset \mathbf{TIME}((2^{(2n+1)})^3)$. But $\mathbf{TIME}((2^{(2n+1)})^3) \subset \mathbf{TIME}(2^{n^2}) \subseteq \mathbf{EXP}$! $\square$

Lemma

If $f(n)$ *is a proper complexity function, then* $\mathbf{SPACE}(f(n))$ *is proper subset of* $\mathbf{SPACE}(f(n) \cdot \log f(n))$.

Theorem (The Gap Theorem)

There exists a recursive function $f : \mathcal{N} \rightarrow \mathcal{N}$ *such that* $\mathbf{TIME}(f(n)) = \mathbf{TIME}(2^{f(n)})$.

Consequences of the Hierarchy Theorem

Lemma

P *is a proper subset of* **EXP**.

Proof.

Clearly **P** $\subseteq$ **TIME**(2^n). As per the Hierarchy theorem, **TIME**$(2^n) \subset$ **TIME**$((2^{(2n+1)})^3)$. But **TIME**$((2^{(2n+1)})^3) \subset$ **TIME**$(2^{n^2}) \subseteq$ **EXP**! $\square$

Lemma

If $f(n)$ *is a proper complexity function, then* **SPACE**$(f(n))$ *is proper subset of* **SPACE**$(f(n) \cdot \log f(n))$.

Theorem (The Gap Theorem)

There exists a recursive function $f : \mathcal{N} \rightarrow \mathcal{N}$ *such that* **TIME**$(f(n)) =$ **TIME**$(2^{f(n)})$.

Consequences of the Hierarchy Theorem

Lemma

P *is a proper subset of* **EXP**.

Proof.

Clearly $\mathbf{P} \subseteq \mathbf{TIME}(2^n)$. As per the Hierarchy theorem, $\mathbf{TIME}(2^n) \subset \mathbf{TIME}((2^{(2n+1)})^3)$. But $\mathbf{TIME}((2^{(2n+1)})^3) \subset \mathbf{TIME}(2^{n^2}) \subseteq \mathbf{EXP}$! $\square$

Lemma

If $f(n)$ *is a proper complexity function, then* $\mathbf{SPACE}(f(n))$ *is proper subset of* $\mathbf{SPACE}(f(n) \cdot \log f(n))$.

Theorem (The Gap Theorem)

There exists a recursive function $f : \mathcal{N} \rightarrow \mathcal{N}$ *such that* $\mathbf{TIME}(f(n)) = \mathbf{TIME}(2^{f(n)})$.