Computational Complexity

K. Subramani[1]

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

16 January, 2014

Outline

Outline

1. The class **NP**
2. The class **coNP**

Outline

Class **NP**

Recall

NP is the class of languages decided by nondeterministic Turing machines in polynomial time.

Class **NP**

Recall

NP is the class of languages decided by nondeterministic Turing machines in polynomial time.

Definition

Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary relation on strings. R is called **polynomially decidable** if the language $\{x; y : (x, y) \in R\}$ is decided by a deterministic Turing machine in polynomial time.

Class **NP**

Recall

NP is the class of languages decided by nondeterministic Turing machines in polynomial time.

Definition

Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary relation on strings. R is called **polynomially decidable** if the language $\{x; y : (x, y) \in R\}$ is decided by a deterministic Turing machine in polynomial time.

Definition

R is **polynomial balanced** if $(x, y) \in R$ implies $|y| \leq |x|^k$ for some $k \geq 1$.
That is, the length of the second component is always bounded by a polynomial in the length of the first.

Class NP (contd.)

Proposition

*Let $L \subseteq \Sigma^*$ be a language. $L \in$ **NP** if and only if there is a polynomial decidable and polynomial balanced relation R, such that $L = \{x : \exists y, (x, y) \in R\}$.*

Class NP (contd.)

Proposition

*Let $L \subseteq \Sigma^*$ be a language. $L \in$ **NP** if and only if there is a polynomial decidable and polynomial balanced relation R, such that $L = \{x : \exists y, (x, y) \in R\}$.*

Proof.

(i) **"If"** part: Suppose that such an R exists, we need to show L is decided by a nondeterministic Turing machine M in polynomial time.

Class NP (contd.)

Proposition

*Let $L \subseteq \Sigma^*$ be a language. $L \in$ **NP** if and only if there is a polynomial decidable and polynomial balanced relation R, such that $L = \{x : \exists y, (x,y) \in R\}$.*

Proof.

(i) **"If"** part: Suppose that such an R exists, we need to show L is decided by a nondeterministic Turing machine M in polynomial time. We construct M as follows: On input x, M guesses a y of length at most $|x|^k$, and then verify whether $(x,y) \in R$ (This can be done in polynomial time because R is polynomial decidable.). If $(x,y) \in R$, M accepts, otherwise it rejects.

Class NP (contd.)

Proposition

*Let $L \subseteq \Sigma^*$ be a language. $L \in$ **NP** if and only if there is a polynomial decidable and polynomial balanced relation R, such that $L = \{x : \exists y, (x, y) \in R\}$.*

Proof.

(i) **"If"** part: Suppose that such an R exists, we need to show L is decided by a nondeterministic Turing machine M in polynomial time. We construct M as follows: On input x, M guesses a y of length at most $|x|^k$, and then verify whether $(x, y) \in R$ (This can be done in polynomial time because R is polynomial decidable.). If $(x, y) \in R$, M accepts, otherwise it rejects.

(ii) **"Only if"** part: Suppose that $L \in$ **NP**, that is, there is a nondeterministic Turing machine N that decides L in time $|x|^k$ for some k.

Class NP (contd.)

Proposition

*Let $L \subseteq \Sigma^*$ be a language. $L \in$ **NP** if and only if there is a polynomial decidable and polynomial balanced relation R, such that $L = \{x : \exists y, (x, y) \in R\}$.*

Proof.

(i) **"If"** part: Suppose that such an R exists, we need to show L is decided by a nondeterministic Turing machine M in polynomial time. We construct M as follows: On input x, M guesses a y of length at most $|x|^k$, and then verify whether $(x, y) \in R$ (This can be done in polynomial time because R is polynomial decidable.). If $(x, y) \in R$, M accepts, otherwise it rejects.

(ii) **"Only if"** part: Suppose that $L \in$ **NP**, that is, there is a nondeterministic Turing machine N that decides L in time $|x|^k$ for some k. Define a relation R as follows: $(x, y) \in R$ if and only if y encodes an accepting computation of N on input x. Clearly R is polynomial decidable and polynomial bounded.

Class NP (contd.)

Proposition

*Let $L \subseteq \Sigma^*$ be a language. $L \in$ **NP** if and only if there is a polynomial decidable and polynomial balanced relation R, such that $L = \{x : \exists y, (x, y) \in R\}$.*

Proof.

(i) **"If"** part: Suppose that such an R exists, we need to show L is decided by a nondeterministic Turing machine M in polynomial time. We construct M as follows: On input x, M guesses a y of length at most $|x|^k$, and then verify whether $(x, y) \in R$ (This can be done in polynomial time because R is polynomial decidable.). If $(x, y) \in R$, M accepts, otherwise it rejects.

(ii) **"Only if"** part: Suppose that $L \in$ **NP**, that is, there is a nondeterministic Turing machine N that decides L in time $|x|^k$ for some k. Define a relation R as follows: $(x, y) \in R$ if and only if y encodes an accepting computation of N on input x. Clearly R is polynomial decidable and polynomial bounded.
Now we show $L = \{x : \exists y, (x, y) \in R\}$.

Class NP (contd.)

Proposition

Let $L \subseteq \Sigma^*$ be a language. $L \in$ **NP** *if and only if there is a polynomial decidable and polynomial balanced relation R, such that $L = \{x : \exists y, (x, y) \in R\}$.*

Proof.

(i) **"If"** part: Suppose that such an R exists, we need to show L is decided by a nondeterministic Turing machine M in polynomial time. We construct M as follows: On input x, M guesses a y of length at most $|x|^k$, and then verify whether $(x, y) \in R$ (This can be done in polynomial time because R is polynomial decidable.). If $(x, y) \in R$, M accepts, otherwise it rejects.

(ii) **"Only if"** part: Suppose that $L \in$ **NP**, that is, there is a nondeterministic Turing machine N that decides L in time $|x|^k$ for some k. Define a relation R as follows: $(x, y) \in R$ if and only if y encodes an accepting computation of N on input x. Clearly R is polynomial decidable and polynomial bounded.
Now we show $L = \{x : \exists y, (x, y) \in R\}$. Since N decides L, $\forall x \in L$, there must be a y such that $(x, y) \in R$, and hence $L \subseteq \{x : \exists y, (x, y) \in R\}$;

Class NP (contd.)

Proposition

*Let $L \subseteq \Sigma^*$ be a language. $L \in$ **NP** if and only if there is a polynomial decidable and polynomial balanced relation R, such that $L = \{x : \exists y, (x, y) \in R\}$.*

Proof.

(i) **"If"** part: Suppose that such an R exists, we need to show L is decided by a nondeterministic Turing machine M in polynomial time. We construct M as follows: On input x, M guesses a y of length at most $|x|^k$, and then verify whether $(x, y) \in R$ (This can be done in polynomial time because R is polynomial decidable.). If $(x, y) \in R$, M accepts, otherwise it rejects.

(ii) **"Only if"** part: Suppose that $L \in$ **NP**, that is, there is a nondeterministic Turing machine N that decides L in time $|x|^k$ for some k. Define a relation R as follows: $(x, y) \in R$ if and only if y encodes an accepting computation of N on input x. Clearly R is polynomial decidable and polynomial bounded.
Now we show $L = \{x : \exists y, (x, y) \in R\}$. Since N decides L, $\forall x \in L$, there must be a y such that $(x, y) \in R$, and hence $L \subseteq \{x : \exists y, (x, y) \in R\}$; Conversely, $\forall x \in \{x : \exists y, (x, y) \in R\}$, it must be the cast that N accepts x. It means $x \in L$, and hence $\{x : \exists y, (x, y) \in R\} \subseteq L$. Thus $L = \{x : \exists y, (x, y) \in R\}$.

$\square$

What does the proposition tell us?

What does the proposition tell us?

Note

(i) *Any "yes" instance x of the problem in* **NP** *has at least one polynomial certificate y of its being a "yes" instance.*

What does the proposition tell us?

Note

(i) *Any "yes" instance x of the problem in* **NP** *has at least one polynomial certificate y of its being a "yes" instance.*

(ii) *We may not know how to discover this certificate in polynomial time, but we are sure it exists if the instance is a "yes" instance.*

What does the proposition tell us?

Note

(i) *Any "yes" instance x of the problem in* **NP** *has at least one polynomial certificate y of its being a "yes" instance.*

(ii) *We may not know how to discover this certificate in polynomial time, but we are sure it exists if the instance is a "yes" instance.*

(iii) *Naturally, "no" instances may not have such certificate.*

What does the proposition tell us?

Note

(i) *Any "yes" instance x of the problem in* **NP** *has at least one polynomial certificate y of its being a "yes" instance.*

(ii) *We may not know how to discover this certificate in polynomial time, but we are sure it exists if the instance is a "yes" instance.*

(iii) *Naturally, "no" instances may not have such certificate.*

Examples

SAT: The certificate is just an assignment that satisfies the Boolean expression.
HAMILTON PATH: The certificate is precisely a Hamilton path in the graph.

SAT

Recall

(**Cook's Theorem**) SAT is **NP**-complete.

SAT

Recall

(**Cook's Theorem**) SAT is **NP**-complete.

Definition

kSAT, where $k \geq 1$ is an integer, is the special case of SAT in which the formula is in CNF, and all clauses have k literals.

coNP as related to NP

Definition (coNP)

coNP is the complexity class which contains the complements of problems found in **NP**.

coNP as related to NP

Definition (coNP)

coNP is the complexity class which contains the complements of problems found in **NP**.

Another way of looking at coNP

Just as **NP** can be considered to be the set of problems with succinct ”yes” certificates, **coNP** can be considered to be the set of problems with succinct ”no” certificates. This means that a ”no” instance of a problem in **coNP** has a short proof of it being a ”no” instance.

A **coNP** problem

Examples

1. coSAT $= \{\langle b \rangle : b$ is a boolean expression with no satisfying assignments$\}$

The class **NP** $\cap$ **coNP**

Properties

Poblems in the class **NP** $\cap$ **coNP** have both succinct “yes” and succinct “no” certificates.

Inclusion Relationships

Relation to P

Just as $\mathbf{P} \subseteq \mathbf{NP}$, we have that $\mathbf{P} = \mathbf{coP} \subseteq \mathbf{coNP}$.

Inclusion Relationships

Relation to P

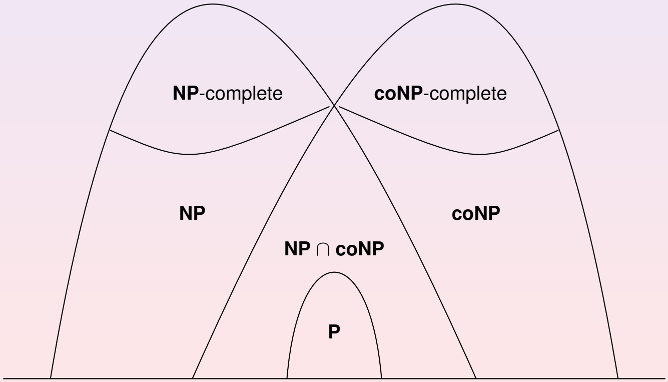
Just as $\mathbf{P} \subseteq \mathbf{NP}$, we have that $\mathbf{P} = \mathbf{coP} \subseteq \mathbf{coNP}$. Thus $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$.

Inclusion Relationships

Relation to P

Just as $\mathbf{P} \subseteq \mathbf{NP}$, we have that $\mathbf{P} = \mathbf{coP} \subseteq \mathbf{coNP}$. Thus $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$.
It is unknown if $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$.

The Complexity Picture



A randomized algorithm for the 2SAT problem

Goal

Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ denote a boolean formula in CNF over the boolean variables $\{x_1, x_2, \ldots, x_n\}$, such that each clause C_i has exactly two variables. Determine whether ϕ is satisfiable.

A randomized algorithm for the 2SAT problem

Goal

Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ denote a boolean formula in CNF over the boolean variables $\{x_1, x_2, \ldots, x_n\}$, such that each clause C_i has exactly two variables. Determine whether ϕ is satisfiable.

Note

2SAT can be solved in $O(m+n)$ time using Tarjan's connected components algorithm.

The 2CNF Algorithm

Function SATISFIABILITY-TESTING(ϕ)

1: Start with an arbitrary assignment to the variables.

The 2CNF Algorithm

Function SATISFIABILITY-TESTING(ϕ)

1: Start with an arbitrary assignment to the variables.
2: **while** (the current assignment is not satisfying) **do**

The 2CNF Algorithm

Function SATISFIABILITY-TESTING(ϕ)

```
Start with an arbitrary assignment to the variables.
while (the current assignment is not satisfying) do
    Pick an unsatisfied clause.
```

The 2CNF Algorithm

Function SATISFIABILITY-TESTING(ϕ)

```
Start with an arbitrary assignment to the variables.
while (the current assignment is not satisfying) do
    Pick an unsatisfied clause.
    Uniformly and at random flip the value assigned to one of its two literals (variables).
```

The 2CNF Algorithm

Function SATISFIABILITY-TESTING(ϕ)

```
Start with an arbitrary assignment to the variables.
while (the current assignment is not satisfying) do
    Pick an unsatisfied clause.
    Uniformly and at random flip the value assigned to one of its two literals (variables).
end while
```

Algorithm 5: Papadimitrious's randomized algorithm for 2CNF Satisfiability

Mathematical Preliminaries

Theorem

Let X and Y be two random variables. Then $\mathbf{E}[X] = \mathbf{E}[\mathbf{E}[X|Y]]$.

Mathematical Preliminaries

Theorem

Let X and Y be two random variables. Then $\mathbf{E}[X] = \mathbf{E}[\mathbf{E}[X|Y]]$.

Theorem (Markov)

Let X be a non-negative random variable and let $c > 0$ denote a constant. Then $\mathbf{Pr}(X \geq c \cdot \mathbf{E}[X]) \leq \frac{1}{c}$.

Analysis

Modeling as a random walk

Assume that ϕ is satisfiable and focus on a particular satisfying assignment $\hat{T}$.

Analysis

Modeling as a random walk

Assume that ϕ is satisfiable and focus on a particular satisfying assignment $\hat{T}$. Let T denote the current assignment. We want to bound the expected number of steps before T is transformed into $\hat{T}$.

Analysis

Modeling as a random walk

Assume that ϕ is satisfiable and focus on a particular satisfying assignment $\hat{T}$. Let T denote the current assignment. We want to bound the expected number of steps before T is transformed into $\hat{T}$.
Let $t(i)$ denote the expected number of flips for T to become $\hat{T}$, assuming that T differs from $\hat{T}$ in exactly i variables.

Analysis

Modeling as a random walk

Assume that ϕ is satisfiable and focus on a particular satisfying assignment $\hat{T}$. Let T denote the current assignment. We want to bound the expected number of steps before T is transformed into $\hat{T}$.
Let $t(i)$ denote the expected number of flips for T to become $\hat{T}$, assuming that T differs from $\hat{T}$ in exactly i variables. It follows that,

$$t(0) =$$

Analysis

Modeling as a random walk

Assume that ϕ is satisfiable and focus on a particular satisfying assignment $\hat{T}$. Let T denote the current assignment. We want to bound the expected number of steps before T is transformed into $\hat{T}$.

Let $t(i)$ denote the expected number of flips for T to become $\hat{T}$, assuming that T differs from $\hat{T}$ in exactly i variables. It follows that,

$$t(0) = 0$$

Analysis

Modeling as a random walk

Assume that ϕ is satisfiable and focus on a particular satisfying assignment $\hat{T}$. Let T denote the current assignment. We want to bound the expected number of steps before T is transformed into $\hat{T}$.

Let $t(i)$ denote the expected number of flips for T to become $\hat{T}$, assuming that T differs from $\hat{T}$ in exactly i variables. It follows that,

$$\begin{aligned} t(0) &= 0 \\ t(n) &= \end{aligned}$$

Analysis

Modeling as a random walk

Assume that ϕ is satisfiable and focus on a particular satisfying assignment $\hat{T}$. Let T denote the current assignment. We want to bound the expected number of steps before T is transformed into $\hat{T}$.
Let $t(i)$ denote the expected number of flips for T to become $\hat{T}$, assuming that T differs from $\hat{T}$ in exactly i variables. It follows that,

$$\begin{aligned} t(0) &= 0 \\ t(n) &= 1 + t(n-1) \end{aligned}$$

Analysis

Modeling as a random walk

Assume that ϕ is satisfiable and focus on a particular satisfying assignment $\hat{T}$. Let T denote the current assignment. We want to bound the expected number of steps before T is transformed into $\hat{T}$.
Let $t(i)$ denote the expected number of flips for T to become $\hat{T}$, assuming that T differs from $\hat{T}$ in exactly i variables. It follows that,

$$\begin{aligned} t(0) &= 0 \\ t(n) &= 1 + t(n-1) \\ t(i) &\leq \end{aligned}$$

Analysis

Modeling as a random walk

Assume that ϕ is satisfiable and focus on a particular satisfying assignment $\hat{T}$. Let T denote the current assignment. We want to bound the expected number of steps before T is transformed into $\hat{T}$.
Let $t(i)$ denote the expected number of flips for T to become $\hat{T}$, assuming that T differs from $\hat{T}$ in exactly i variables. It follows that,

$$\begin{aligned} t(0) &= 0 \\ t(n) &= 1 + t(n-1) \\ t(i) &\leq \frac{1}{2}t(i-1) + \frac{1}{2}t(i+1) + 1, 0 < i < n \end{aligned}$$

Analysis

Modeling as a random walk

Assume that ϕ is satisfiable and focus on a particular satisfying assignment $\hat{T}$. Let T denote the current assignment. We want to bound the expected number of steps before T is transformed into $\hat{T}$.
Let $t(i)$ denote the expected number of flips for T to become $\hat{T}$, assuming that T differs from $\hat{T}$ in exactly i variables. It follows that,

$$\begin{aligned} t(0) &= 0 \\ t(n) &= 1 + t(n-1) \\ t(i) &\leq \frac{1}{2}t(i-1) + \frac{1}{2}t(i+1) + 1, 0 < i < n \end{aligned}$$

Observation

The above system can be solved to get $t(n) \leq n^2$. From Markov's inequality it follows that the probability that T is not transformed into $\hat{T}$ in at most $2 \cdot n^2$ flips is less than one-half.

Analysis

Modeling as a random walk

Assume that ϕ is satisfiable and focus on a particular satisfying assignment $\hat{T}$. Let T denote the current assignment. We want to bound the expected number of steps before T is transformed into $\hat{T}$.
Let $t(i)$ denote the expected number of flips for T to become $\hat{T}$, assuming that T differs from $\hat{T}$ in exactly i variables. It follows that,

$$\begin{aligned} t(0) &= 0 \\ t(n) &= 1 + t(n-1) \\ t(i) &\leq \frac{1}{2}t(i-1) + \frac{1}{2}t(i+1) + 1, 0 < i < n \end{aligned}$$

Observation

The above system can be solved to get $t(n) \leq n^2$. From Markov's inequality it follows that the probability that T is not transformed into $\hat{T}$ in at most $2 \cdot n^2$ flips is less than one-half. Running time is $O(n^2 \cdot (m+n))$, which is hardly impressive.

Randomized Complexity Classes

Randomized Complexity Classes

Definition

The class **RP** consists of all languages $L \subseteq \Sigma^*$ that have a randomized algorithm $\mathscr{A}$ running in worst-case polynomial time, such that for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{``}yes\text{''}] \geq \frac{1}{2}$.
- $x \notin L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{``}yes\text{''}] = 0$.

Observations

(i) *Rejection is unanimous, acceptance is by majority.*

Randomized Complexity Classes

Definition

The class **RP** consists of all languages $L \subseteq \Sigma^*$ that have a randomized algorithm $\mathscr{A}$ running in worst-case polynomial time, such that for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{“}yes\text{''}] \geq \frac{1}{2}$.
- $x \notin L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{“}yes\text{''}] = 0$.

Observations

(i) *Rejection is unanimous, acceptance is by majority.*

(ii) *Only positive-sided error is allowed.*

Randomized Complexity Classes

Definition

The class **RP** consists of all languages $L \subseteq \Sigma^*$ that have a randomized algorithm $\mathscr{A}$ running in worst-case polynomial time, such that for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{``}yes\text{''}] \geq \frac{1}{2}$.
- $x \notin L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{``}yes\text{''}] = 0$.

Observations

(i) *Rejection is unanimous, acceptance is by majority.*

(ii) *Only positive-sided error is allowed.*

(iii) *The number $\frac{1}{2}$ can be any fixed constant between 0 and 1, without affecting the set of languages in* **RP**.

Randomized Complexity Classes

Definition

The class **RP** consists of all languages $L \subseteq \Sigma^*$ that have a randomized algorithm $\mathscr{A}$ running in worst-case polynomial time, such that for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{``}yes\text{''}] \geq \frac{1}{2}$.
- $x \notin L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{``}yes\text{''}] = 0$.

Observations

(i) *Rejection is unanimous, acceptance is by majority.*

(ii) *Only positive-sided error is allowed.*

(iii) *The number $\frac{1}{2}$ can be any fixed constant between 0 and 1, without affecting the set of languages in* **RP**.

(iv) *2SAT is in* **RP**.

Randomized Complexity Classes (contd.)

Definition

A language $L \subseteq \Sigma^*$ is in **coRP**, if its complement is in **RP**.

Randomized Complexity Classes (contd.)

Definition

A language $L \subseteq \Sigma^*$ is in **coRP**, if its complement is in **RP**.

Definition

A language $L \subseteq \Sigma^*$ is in **ZPP** is it is in **RP** $\cap$ **coRP**.

Randomized Complexity Classes (contd.)

Definition

A language $L \subseteq \Sigma^*$ is in **BPP**, if there exists a randomized algorithm $\mathscr{A}$ running in worst-case polynomial time, such that for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{“}yes\text{”}] \geq \frac{3}{4}$.

Randomized Complexity Classes (contd.)

Definition

A language $L \subseteq \Sigma^*$ is in **BPP**, if there exists a randomized algorithm $\mathscr{A}$ running in worst-case polynomial time, such that for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{"}yes\text{"}] \geq \frac{3}{4}$.
- $x \notin L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{"}yes\text{"}] \leq \frac{1}{4}$.

Randomized Complexity Classes (contd.)

Definition

A language $L \subseteq \Sigma^*$ is in **BPP**, if there exists a randomized algorithm $\mathscr{A}$ running in worst-case polynomial time, such that for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{"}yes\text{"}] \geq \frac{3}{4}$.
- $x \notin L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{"}yes\text{"}] \leq \frac{1}{4}$.

Alternative view of **RP**

RP denotes the set of languages L which can be decided by a polynomially bounded non-deterministic Turing machine N in the following manner: For each input x, if $x \in L$, then at least half the computations of N on x end in accepting leaves and if $x \notin L$, the all computations of N on x end in rejecting leaves.

Randomized Complexity Classes (contd.)

Definition

A language $L \subseteq \Sigma^*$ is in **BPP**, if there exists a randomized algorithm $\mathscr{A}$ running in worst-case polynomial time, such that for any input $x \in \Sigma^*$,

- $x \in L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{“}yes\text{''}] \geq \frac{3}{4}$.
- $x \notin L \Rightarrow \mathbf{Pr}[\mathscr{A}(x)] = \text{“}yes\text{''}] \leq \frac{1}{4}$.

Alternative view of **RP**

RP denotes the set of languages L which can be decided by a polynomially bounded non-deterministic Turing machine N in the following manner: For each input x, if $x \in L$, then at least half the computations of N on x end in accepting leaves and if $x \notin L$, the all computations of N on x end in rejecting leaves. WIthout loss of generality, we may assume that the degree of non-determinism is exactly 2 at each node of the computation tree.

Relations between complexity classes

Observations

Relations between complexity classes

Observations

(i) $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$.

Relations between complexity classes

Observations

(i) $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$.

(ii) $\mathbf{P} \subseteq \mathbf{coRP} \subseteq \mathbf{coNP}$.

Relations between complexity classes

Observations

(i) $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$.

(ii) $\mathbf{P} \subseteq \mathbf{coRP} \subseteq \mathbf{coNP}$.

(iii) $\mathbf{RP} \subseteq \mathbf{BPP}$.

The Complexity Picture