The polynomial hierarchy and PSPACE

Piotr Wojciechowski[1]

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

Outline

1. The complexity class DP
   - Definition of DP
   - Problems in DP
2. The classes P^{NP} and FP^{NP}
   - The definition of P^{NP} and FP^{NP}
3. The polynomial Hierarchy
   - The definition of the Polynomial Hierarchy
   - Examining the Polynomial Hierarchy
   - Diagram of the complexity classes
4. A look a PSPACE
   - QSAT is PSPACE complete
   - PSPACE=AP
   - Geography is PSPACE-complete

Outline

1. The complexity class DP
   - Definition of DP
   - Problems in DP
2. The classes P^{NP} and FP^{NP}
   - The definition of P^{NP} and FP^{NP}
3. The polynomial Hierarchy
   - The definition of the Polynomial Hierarchy
   - Examining the Polynomial Hierarchy
   - Diagram of the complexity classes
4. A look a PSPACE
   - QSAT is PSPACE complete
   - PSPACE=AP
   - Geography is PSPACE-complete

Outline

Outline

Outline

Definition (DP)

A Language L is in the class DP if and only if there are two languages $L_1 \in$ NP and $L_2 \in$ coNP such that $L = L_1 \cap L_2$.

Outline

SAT-UNSAT

Definition (SAT-UNSAT)

Given two boolean clauses ϕ, ϕ' both in conjunctive normal from with three literals per clause. Is it true that ϕ is satisfiable and ϕ' is not.

Theorem

SAT-UNSAT is DP-complete.

Proof.

1. SAT-UNSAT $\in$ DP.
   Simple let $L_1 = \{(\phi, \phi') : \phi \text{ is satisfiable}\}$ and $L_2 = \{(\phi, \phi') : \phi' \text{ is unsatisfiable}\}$
2. If $L \in$ DP then L reduces to SAT-UNSAT
   Let $L_1 \in$ NP and $L_2 \in$ coNP be languages such that $L = L_1 \cap L_2$. Let R_1 be a reduction from L_1 to SAT and let R_2 be a reduction from L_2 to UNSAT. Thus the reduction R from L to SAT-UNSAT is for input x, $R(x) = (R_1(x), R_2(x))$.
   We have that $R(x) \in$ SAT-UNSAT iff $R_1(x) \in$ SAT and $R_2(x) \in$ UNSAT, which is true iff $x \in L_1$ and $x \in L_2$, or equivalently that $x \in L$.

□

SAT-UNSAT

Definition (SAT-UNSAT)

Given two boolean clauses ϕ, ϕ' both in conjunctive normal from with three literals per clause. Is it true that ϕ is satisfiable and ϕ' is not.

Theorem

SAT-UNSAT is DP-complete.

Proof.

1. SAT-UNSAT $\in$ DP.
   Simple let $L_1 = \{(\phi, \phi') : \phi \text{ is satisfiable}\}$ and $L_2 = \{(\phi, \phi') : \phi' \text{ is unsatisfiable}\}$
2. If $L \in$ DP then L reduces to SAT-UNSAT
   Let $L_1 \in$ NP and $L_2 \in$ coNP be languages such that $L = L_1 \cap L_2$. Let R_1 be a reduction from L_1 to SAT and let R_2 be a reduction from L_2 to UNSAT. Thus the reduction R from L to SAT-UNSAT is for input x, $R(x) = (R_1(x), R_2(x))$.
   We have that $R(x) \in$ SAT-UNSAT iff $R_1(x) \in$ SAT and $R_2(x) \in$ UNSAT, which is true iff $x \in L_1$ and $x \in L_2$, or equivalently that $x \in L$.

□

SAT-UNSAT

Definition (SAT-UNSAT)

Given two boolean clauses ϕ, ϕ' both in conjunctive normal from with three literals per clause. Is it true that ϕ is satisfiable and ϕ' is not.

Theorem

SAT-UNSAT is DP-complete.

Proof.

1. SAT-UNSAT $\in$ DP.
   Simple let $L_1 = \{(\phi, \phi') : \phi \text{ is satisfiable}\}$ and $L_2 = \{(\phi, \phi') : \phi' \text{ is unsatisfiable}\}$
2. If $L \in$ DP then L reduces to SAT-UNSAT
   Let $L_1 \in$ NP and $L_2 \in$ coNP be languages such that $L = L_1 \cap L_2$. Let R_1 be a reduction from L_1 to SAT and let R_2 be a reduction from L_2 to UNSAT. Thus the reduction R from L to SAT-UNSAT is for input x, $R(x) = (R_1(x), R_2(x))$.
   We have that $R(x) \in$ SAT-UNSAT iff $R_1(x) \in$ SAT and $R_2(x) \in$ UNSAT, which is true iff $x \in L_1$ and $x \in L_2$, or equivalently that $x \in L$.

□

SAT-UNSAT

Definition (SAT-UNSAT)

Given two boolean clauses ϕ, ϕ' both in conjunctive normal from with three literals per clause. Is it true that ϕ is satisfiable and ϕ' is not.

Theorem

SAT-UNSAT is DP-complete.

Proof.

1. SAT-UNSAT $\in$ DP.
   Simple let $L_1 = \{(\phi, \phi') : \phi \text{ is satisfiable}\}$ and $L_2 = \{(\phi, \phi') : \phi' \text{ is unsatisfiable}\}$
2. If $L \in$ DP then L reduces to SAT-UNSAT
   Let $L_1 \in$ NP and $L_2 \in$ coNP be languages such that $L = L_1 \cap L_2$. Let R_1 be a reduction from L_1 to SAT and let R_2 be a reduction from L_2 to UNSAT. Thus the reduction R from L to SAT-UNSAT is for input x, $R(x) = (R_1(x), R_2(x))$.
   We have that $R(x) \in$ SAT-UNSAT iff $R_1(x) \in$ SAT and $R_2(x) \in$ UNSAT, which is true iff $x \in L_1$ and $x \in L_2$, or equivalently that $x \in L$.

SAT-UNSAT

Definition (SAT-UNSAT)

Given two boolean clauses ϕ, ϕ' both in conjunctive normal from with three literals per clause. Is it true that ϕ is satisfiable and ϕ' is not.

Theorem

SAT-UNSAT is DP-complete.

Proof.

1. SAT-UNSAT $\in$ DP.
   Simple let $L_1 = \{(\phi, \phi') : \phi \text{ is satisfiable}\}$ and $L_2 = \{(\phi, \phi') : \phi' \text{ is unsatisfiable}\}$
2. If $L \in$ DP then L reduces to SAT-UNSAT
   Let $L_1 \in$ NP and $L_2 \in$ coNP be languages such that $L = L_1 \cap L_2$. Let R_1 be a reduction from L_1 to SAT and let R_2 be a reduction from L_2 to UNSAT. Thus the reduction R from L to SAT-UNSAT is for input x, $R(x) = (R_1(x), R_2(x))$.
   We have that $R(x) \in$ SAT-UNSAT iff $R_1(x) \in$ SAT and $R_2(x) \in$ UNSAT, which is true iff $x \in L_1$ and $x \in L_2$, or equivalently that $x \in L$.

□

SAT-UNSAT

Definition (SAT-UNSAT)

Given two boolean clauses ϕ, ϕ' both in conjunctive normal from with three literals per clause. Is it true that ϕ is satisfiable and ϕ' is not.

Theorem

SAT-UNSAT is DP-complete.

Proof.

1. SAT-UNSAT $\in$ DP.
   Simple let $L_1 = \{(\phi, \phi') : \phi \text{ is satisfiable}\}$ and $L_2 = \{(\phi, \phi') : \phi' \text{ is unsatisfiable}\}$
2. If $L \in$ DP then L reduces to SAT-UNSAT
   Let $L_1 \in$ NP and $L_2 \in$ coNP be languages such that $L = L_1 \cap L_2$. Let R_1 be a reduction from L_1 to SAT and let R_2 be a reduction from L_2 to UNSAT. Thus the reduction R from L to SAT-UNSAT is for input x, $R(x) = (R_1(x), R_2(x))$.
   We have that $R(x) \in$ SAT-UNSAT iff $R_1(x) \in$ SAT and $R_2(x) \in$ UNSAT, which is true iff $x \in L_1$ and $x \in L_2$, or equivalently that $x \in L$.

□

SAT-UNSAT

Definition (SAT-UNSAT)

Given two boolean clauses ϕ, ϕ' both in conjunctive normal from with three literals per clause. Is it true that ϕ is satisfiable and ϕ' is not.

Theorem

SAT-UNSAT is DP-complete.

Proof.

1. SAT-UNSAT $\in$ DP.
   Simple let $L_1 = \{(\phi, \phi') : \phi \text{ is satisfiable}\}$ and $L_2 = \{(\phi, \phi') : \phi' \text{ is unsatisfiable}\}$
2. If $L \in$ DP then L reduces to SAT-UNSAT
   Let $L_1 \in$ NP and $L_2 \in$ coNP be languages such that $L = L_1 \cap L_2$. Let R_1 be a reduction from L_1 to SAT and let R_2 be a reduction from L_2 to UNSAT. Thus the reduction R from L to SAT-UNSAT is for input x, $R(x) = (R_1(x), R_2(x))$.
   We have that $R(x) \in$ SAT-UNSAT iff $R_1(x) \in$ SAT and $R_2(x) \in$ UNSAT, which is true iff $x \in L_1$ and $x \in L_2$, or equivalently that $x \in L$.

□

Other problems in DP

Problems

1. EXACT TSP
   Given a distance matrix and an integer B, is the length of the shortest tour *equal* to B.
2. CRITICAL SAT
   Given a boolean expression ϕ is it unsatisfiable, but does removing any clause make it satisfiable
3. CRITICAL HAMILTONIAN PATH
   Given a Graph does it have no Hamiltonian path, but does the addition of any edge give it a Hamiltonian Path.
4. CRITICAL 3-COLORABILITY
   Given a graph is it not three colorable, but does removing any node make it three colorable.

In fact all of these problems are DP-complete

Other problems in DP

Problems

1. EXACT TSP
   Given a distance matrix and an integer B, is the length of the shortest tour *equal* to B.
2. CRITICAL SAT
   Given a boolean expression ϕ is it unsatisfiable, but does removing any clause make it satisfiable
3. CRITICAL HAMILTONIAN PATH
   Given a Graph does it have no Hamiltonian path, but does the addition of any edge give it a Hamiltonian Path.
4. CRITICAL 3-COLORABILITY
   Given a graph is it not three colorable, but does removing any node make it three colorable.

In fact all of these problems are DP-complete

Other problems in DP

Problems

1. EXACT TSP
   Given a distance matrix and an integer B, is the length of the shortest tour *equal* to B.
2. CRITICAL SAT
   Given a boolean expression ϕ is it unsatisfiable, but does removing any clause make it satisfiable
3. CRITICAL HAMILTONIAN PATH
   Given a Graph does it have no Hamiltonian path, but does the addition of any edge give it a Hamiltonian Path.
4. CRITICAL 3-COLORABILITY
   Given a graph is it not three colorable, but does removing any node make it three colorable.

In fact all of these problems are DP-complete

Other problems in DP

Problems

1. EXACT TSP
   Given a distance matrix and an integer B, is the length of the shortest tour *equal* to B.
2. CRITICAL SAT
   Given a boolean expression ϕ is it unsatisfiable, but does removing any clause make it satisfiable
3. CRITICAL HAMILTONIAN PATH
   Given a Graph does it have no Hamiltonian path, but does the addition of any edge give it a Hamiltonian Path.
4. CRITICAL 3-COLORABILITY
   Given a graph is it not three colorable, but does removing any node make it three colorable.

In fact all of these problems are DP-complete

Other problems in DP

Problems

1. EXACT TSP
   Given a distance matrix and an integer B, is the length of the shortest tour *equal* to B.
2. CRITICAL SAT
   Given a boolean expression ϕ is it unsatisfiable, but does removing any clause make it satisfiable
3. CRITICAL HAMILTONIAN PATH
   Given a Graph does it have no Hamiltonian path, but does the addition of any edge give it a Hamiltonian Path.
4. CRITICAL 3-COLORABILITY
   Given a graph is it not three colorable, but does removing any node make it three colorable.

In fact all of these problems are DP-complete

Outline

Definition (P^{NP})

A language L, is in P^{NP} if there exists a language $L' \in$ NP such that L can be decided by a polynomial time Oracle machine using an L' Oracle.

Explanation of an Oracle

A Turing Machine M^A with oracle A is a multi-string Turing Machine with a special string called the *Query String* and special states $q_?$, the *query state*, and q_{YES},q_{NO}, the *answer states*. From $q_?$ M^A moves to q_{YES} or q_{no} depending on whether the query string is in A or not. This result can be used in further computations.

Definition (P^{NP})

A language L, is in P^{NP} if there exists a language $L' \in$ NP such that L can be decided by a polynomial time Oracle machine using an L' Oracle.

Explanation of an Oracle

A Turing Machine M^A with oracle A is a multi-string Turing Machine with a special string called the *Query String* and special states $q_?$, the *query state*, and q_{YES},q_{NO}, the *answer states*. From $q_?$ M^A moves to q_{YES} or q_{no} depending on whether the query string is in A or not. This result can be used in further computations.

Definition (P^{NP})

A language L, is in P^{NP} if there exists a language $L' \in$ NP such that L can be decided by a polynomial time Oracle machine using an L' Oracle.

Explanation of an Oracle

A Turing Machine M^A with oracle A is a multi-string Turing Machine with a special string called the *Query String* and special states $q_?$, the *query state*, and q_{YES},q_{NO}, the *answer states*. From $q_?$ M^A moves to q_{YES} or q_{no} depending on whether the query string is in A or not. This result can be used in further computations.

Theorem

$DP \subseteq P^{NP}$

Proof.

Let $L \in$ DP. We have that L can be reduced in polynomial time to SAT-UNSAT using the reduction shown before. Now simply query whether $R_1(x) \in$ SAT and whether $R_2(x) \notin$ SAT. Where R_1, R_2, and x have the same meanings they did in the reduction. □

Definition (FP^{NP})

FP^{NP} is the set of all function problems that can be computed in polynomial time using an oracle in NP.

Theorem

$DP \subseteq P^{NP}$

Proof.

Let $L \in$ DP. We have that L can be reduced in polynomial time to SAT-UNSAT using the reduction shown before. Now simply query whether $R_1(x) \in$ SAT and whether $R_2(x) \notin$ SAT. Where R_1, R_2, and x have the same meanings they did in the reduction. □

Definition (FP^{NP})

FP^{NP} is the set of all function problems that can be computed in polynomial time using an oracle in NP.

Theorem

$DP \subseteq P^{NP}$

Proof.

Let $L \in$ DP. We have that L can be reduced in polynomial time to SAT-UNSAT using the reduction shown before. Now simply query whether $R_1(x) \in$ SAT and whether $R_2(x) \notin$ SAT. Where R_1, R_2, and x have the same meanings they did in the reduction. □

Definition (FP^{NP})

FP^{NP} is the set of all function problems that can be computed in polynomial time using an oracle in NP.

Theorem

$DP \subseteq P^{NP}$

Proof.

Let $L \in$ DP. We have that L can be reduced in polynomial time to SAT-UNSAT using the reduction shown before. Now simply query whether $R_1(x) \in$ SAT and whether $R_2(x) \notin$ SAT. Where R_1, R_2, and x have the same meanings they did in the reduction. □

Definition (FP^{NP})

FP^{NP} is the set of all function problems that can be computed in polynomial time using an oracle in NP.

Outline

1. The complexity class DP
   - Definition of DP
   - Problems in DP
2. The classes P^{NP} and FP^{NP}
   - The definition of P^{NP} and FP^{NP}
3. The polynomial Hierarchy
   - The definition of the Polynomial Hierarchy
   - Examining the Polynomial Hierarchy
   - Diagram of the complexity classes
4. A look a PSPACE
   - QSAT is PSPACE complete
   - PSPACE=AP
   - Geography is PSPACE-complete

Definition (Polynomial Hierarchy)

The *polynomial hierarchy* is the following sequence of classes:

1. $\Delta_0 P = \Sigma_0 P = \Pi_0 P = P$
2. $\Delta_{i+1} P = P^{\Sigma_i P}$
3. $\Sigma_{i+1} P = NP^{\Sigma_i P}$
4. $\Pi_{i+1} P = coNP^{\Sigma_i P}$

For all $i \geq 0$.
We also define the collective class **PH** $= \bigcup_{i \geq 0} \Sigma_i P$.

Observations

Note that because $\Sigma_0 P = P$, we have that $\Sigma_1 P = NP$, $\Delta_1 P = P$, and $\Pi_1 P = coNP$. At each level the classes are believed to be distinct and are known to hold the same relationship as P, NP and coNP. Also, each class at each level includes all classes at the previous levels.

Definition (Polynomial Hierarchy)

The *polynomial hierarchy* is the following sequence of classes:

1. $\Delta_0 P = \Sigma_0 P = \Pi_0 P = P$
2. $\Delta_{i+1} P = P^{\Sigma_i P}$
3. $\Sigma_{i+1} P = NP^{\Sigma_i P}$
4. $\Pi_{i+1} P = coNP^{\Sigma_i P}$

For all $i \geq 0$.
We also define the collective class **PH** $= \bigcup_{i \geq 0} \Sigma_i P$.

Observations

Note that because $\Sigma_0 P = P$, we have that $\Sigma_1 P = NP$, $\Delta_1 P = P$, and $\Pi_1 P = coNP$. At each level the classes are believed to be distinct and are known to hold the same relationship as P, NP and coNP. Also, each class at each level includes all classes at the previous levels.

Definition (Polynomial Hierarchy)

The *polynomial hierarchy* is the following sequence of classes:

1. $\Delta_0 P = \Sigma_0 P = \Pi_0 P = P$
2. $\Delta_{i+1} P = P^{\Sigma_i P}$
3. $\Sigma_{i+1} P = NP^{\Sigma_i P}$
4. $\Pi_{i+1} P = coNP^{\Sigma_i P}$

For all $i \geq 0$.
We also define the collective class **PH** $= \bigcup_{i \geq 0} \Sigma_i P$.

Observations

Note that because $\Sigma_0 P = P$, we have that $\Sigma_1 P = NP$, $\Delta_1 P = P$, and $\Pi_1 P = coNP$. At each level the classes are believed to be distinct and are known to hold the same relationship as P, NP and coNP. Also, each class at each level includes all classes at the previous levels.

Definition (Polynomial Hierarchy)

The *polynomial hierarchy* is the following sequence of classes:

1. $\Delta_0\mathrm{P} = \Sigma_0\mathrm{P} = \Pi_0\mathrm{P} = \mathrm{P}$
2. $\Delta_{i+1}\mathrm{P} = P^{\Sigma_i P}$
3. $\Sigma_{i+1}\mathrm{P} = NP^{\Sigma_i P}$
4. $\Pi_{i+1}\mathrm{P} = coNP^{\Sigma_i P}$

For all $i \geq 0$.
We also define the collective class $\mathbf{PH} = \bigcup_{i \geq 0} \Sigma_i P$.

Observations

Note that because $\Sigma_0\mathrm{P} = \mathrm{P}$, we have that $\Sigma_1\mathrm{P} = \mathrm{NP}$, $\Delta_1\mathrm{P} = \mathrm{P}$, and $\Pi_1\mathrm{P} = \mathrm{coNP}$. At each level the classes are believed to be distinct and are known to hold the same relationship as P, NP and coNP. Also, each class at each level includes all classes at the previous levels.

Definition (Polynomial Hierarchy)

The *polynomial hierarchy* is the following sequence of classes:

1. $\Delta_0 P = \Sigma_0 P = \Pi_0 P = P$
2. $\Delta_{i+1} P = P^{\Sigma_i P}$
3. $\Sigma_{i+1} P = NP^{\Sigma_i P}$
4. $\Pi_{i+1} P = coNP^{\Sigma_i P}$

For all $i \geq 0$.
We also define the collective class **PH** $= \bigcup_{i \geq 0} \Sigma_i P$.

Observations

Note that because $\Sigma_0 P = P$, we have that $\Sigma_1 P = NP$, $\Delta_1 P = P$, and $\Pi_1 P = coNP$. At each level the classes are believed to be distinct and are known to hold the same relationship as P, NP and coNP. Also, each class at each level includes all classes at the previous levels.

Definition (Polynomial Hierarchy)

The *polynomial hierarchy* is the following sequence of classes:

1. $\Delta_0 P = \Sigma_0 P = \Pi_0 P = P$
2. $\Delta_{i+1} P = P^{\Sigma_i P}$
3. $\Sigma_{i+1} P = NP^{\Sigma_i P}$
4. $\Pi_{i+1} P = coNP^{\Sigma_i P}$

For all $i \geq 0$.
We also define the collective class **PH** $= \bigcup_{i \geq 0} \Sigma_i P$.

Observations

Note that because $\Sigma_0 P = P$, we have that $\Sigma_1 P = NP$, $\Delta_1 P = P$, and $\Pi_1 P = coNP$. At each level the classes are believed to be distinct and are known to hold the same relationship as P, NP and coNP. Also, each class at each level includes all classes at the previous levels.

Definition (Polynomial Hierarchy)

The *polynomial hierarchy* is the following sequence of classes:

1. Δ_0P = Σ_0P = Π_0P = P
2. Δ_{i+1}P = $P^{\Sigma_i P}$
3. Σ_{i+1}P = $NP^{\Sigma_i P}$
4. Π_{i+1}P = $coNP^{\Sigma_i P}$

For all $i \geq 0$.
We also define the collective class **PH** $= \bigcup_{i \geq 0} \Sigma_i P$.

Observations

Note that because Σ_0P = P, we have that Σ_1P = NP, Δ_1P = P, and Π_1P = coNP. At each level the classes are believed to be distinct and are known to hold the same relationship as P, NP and coNP. Also, each class at each level includes all classes at the previous levels.

Definition (Polynomial Hierarchy)

The *polynomial hierarchy* is the following sequence of classes:

1. $\Delta_0 \mathrm{P} = \Sigma_0 \mathrm{P} = \Pi_0 \mathrm{P} = \mathrm{P}$
2. $\Delta_{i+1}\mathrm{P} = P^{\Sigma_i P}$
3. $\Sigma_{i+1}\mathrm{P} = NP^{\Sigma_i P}$
4. $\Pi_{i+1}\mathrm{P} = coNP^{\Sigma_i P}$

For all $i \geq 0$.
We also define the collective class **PH** $= \bigcup_{i \geq 0} \Sigma_i P$.

Observations

Note that because $\Sigma_0\mathrm{P} = \mathrm{P}$, we have that $\Sigma_1\mathrm{P} = \mathrm{NP}$, $\Delta_1\mathrm{P} = \mathrm{P}$, and $\Pi_1\mathrm{P} = \mathrm{coNP}$. At each level the classes are believed to be distinct and are known to hold the same relationship as P, NP and coNP. Also, each class at each level includes all classes at the previous levels.

Definition (Polynomial Hierarchy)

The *polynomial hierarchy* is the following sequence of classes:

1. $\Delta_0\text{P} = \Sigma_0\text{P} = \Pi_0\text{P} = \text{P}$
2. $\Delta_{i+1}\text{P} = P^{\Sigma_i P}$
3. $\Sigma_{i+1}\text{P} = NP^{\Sigma_i P}$
4. $\Pi_{i+1}\text{P} = coNP^{\Sigma_i P}$

For all $i \geq 0$.
We also define the collective class **PH** $= \bigcup_{i \geq 0} \Sigma_i P$.

Observations

Note that because $\Sigma_0\text{P} = \text{P}$, we have that $\Sigma_1\text{P} = \text{NP}$, $\Delta_1\text{P} = \text{P}$, and $\Pi_1\text{P} = \text{coNP}$. At each level the classes are believed to be distinct and are known to hold the same relationship as P, NP and coNP. Also, each class at each level includes all classes at the previous levels.

Definition (Polynomial Hierarchy)

The *polynomial hierarchy* is the following sequence of classes:

1. $\Delta_0\mathrm{P} = \Sigma_0\mathrm{P} = \Pi_0\mathrm{P} = \mathrm{P}$
2. $\Delta_{i+1}\mathrm{P} = P^{\Sigma_i P}$
3. $\Sigma_{i+1}\mathrm{P} = NP^{\Sigma_i P}$
4. $\Pi_{i+1}\mathrm{P} = coNP^{\Sigma_i P}$

For all $i \geq 0$.
We also define the collective class **PH** $= \bigcup_{i\geq 0} \Sigma_i P$.

Observations

Note that because $\Sigma_0\mathrm{P} = \mathrm{P}$, we have that $\Sigma_1\mathrm{P} = \mathrm{NP}$, $\Delta_1\mathrm{P} = \mathrm{P}$, and $\Pi_1\mathrm{P} = \mathrm{coNP}$. At each level the classes are believed to be distinct and are known to hold the same relationship as P, NP and coNP. Also, each class at each level includes all classes at the previous levels.

Outline

Theorem

Let L be a Language, and $i \geq 1$. $L \in \Sigma_i P$ iff there is a polynomially balanced relation R such that the language $\{(x, y) : (x, y) \in R\}$ is in $\Pi_{i-1}P$ and $L = \{x$: there is a y such that $(x, y) \in R\}$.

Proof.

We will show this by using induction on i.
If $i = 1$ then this reduces to proposition 9.1.
If $i > 1$ then suppose such a relation R exists, to show that $L \in \Sigma_i$P we will construct machine M which guessed an appropriate y and asks a Σ_{i-1}P oracle whether $(x, y) \notin R$.
Conversely we can assume that $L \in \Sigma_i$P. By the definition of Σ_iP there is a NDTM M^K using oracle $K \in \Sigma_{i-1}$P. Thus, by induction, there is a relation S recognizable in Π_{i-2}P such that $z \in K$ iff $\exists w$ such that $(z, w) \in S$. □

Theorem

Let L be a Language, and $i \geq 1$*.* $L \in \Sigma_i P$ *iff there is a polynomially balanced relation R such that the language* $\{(x, y) : (x, y) \in R\}$ *is in* $\Pi_{i-1} P$ *and* $L = \{x$*: there is a y such that* $(x, y) \in R\}$*.*

Proof.

We will show this by using induction on i.

If $i = 1$ then this reduces to proposition 9.1.

If $i > 1$ then suppose such a relation R exists, to show that $L \in \Sigma_i$P we will construct machine M which guessed an appropriate y and asks a Σ_{i-1}P oracle whether $(x, y) \notin R$.

Conversely we can assume that $L \in \Sigma_i$P. By the definition of Σ_iP there is a NDTM M^K using oracle $K \in \Sigma_{i-1}$P. Thus, by induction, there is a relation S recognizable in Π_{i-2}P such that $z \in K$ iff $\exists w$ such that $(z, w) \in S$. □

Theorem

Let L be a Language, and $i \geq 1$. $L \in \Sigma_i P$ iff there is a polynomially balanced relation R such that the language $\{(x, y) : (x, y) \in R\}$ is in $\Pi_{i-1} P$ and $L = \{x$: there is a y such that $(x, y) \in R\}$.

Proof.

We will show this by using induction on i.
If $i = 1$ then this reduces to proposition 9.1.
If $i > 1$ then suppose such a relation R exists, to show that $L \in \Sigma_i$P we will construct machine M which guessed an appropriate y and asks a Σ_{i-1}P oracle whether $(x, y) \notin R$.
Conversely we can assume that $L \in \Sigma_i$P. By the definition of Σ_iP there is a NDTM M^K using oracle $K \in \Sigma_{i-1}$P. Thus, by induction, there is a relation S recognizable in Π_{i-2}P such that $z \in K$ iff $\exists w$ such that $(z, w) \in S$. □

Theorem

Let L be a Language, and $i \geq 1$. $L \in \Sigma_i P$ iff there is a polynomially balanced relation R such that the language $\{(x, y) : (x, y) \in R\}$ is in $\Pi_{i-1}P$ and $L = \{x$: there is a y such that $(x, y) \in R\}$.

Proof.

We will show this by using induction on i.
If $i = 1$ then this reduces to proposition 9.1.
If $i > 1$ then suppose such a relation R exists, to show that $L \in \Sigma_i$P we will construct machine M which guessed an appropriate y and asks a Σ_{i-1}P oracle whether $(x, y) \notin R$.
Conversely we can assume that $L \in \Sigma_i$P. By the definition of Σ_iP there is a NDTM M^K using oracle $K \in \Sigma_{i-1}$P. Thus, by induction, there is a relation S recognizable in Π_{i-2}P such that $z \in K$ iff $\exists w$ such that $(z, w) \in S$. □

Theorem

Let L be a Language, and $i \geq 1$. $L \in \Sigma_i P$ iff there is a polynomially balanced relation R such that the language $\{(x, y) : (x, y) \in R\}$ is in $\Pi_{i-1}P$ and $L = \{x$: there is a y such that $(x, y) \in R\}$.

Proof.

We will show this by using induction on i.
If $i = 1$ then this reduces to proposition 9.1.
If $i > 1$ then suppose such a relation R exists, to show that $L \in \Sigma_i$P we will construct machine M which guessed an appropriate y and asks a Σ_{i-1}P oracle whether $(x, y) \notin R$.
Conversely we can assume that $L \in \Sigma_i$P. By the definition of Σ_iP there is a NDTM M^K using oracle $K \in \Sigma_{i-1}$P. Thus, by induction, there is a relation S recognizable in Π_{i-2}P such that $z \in K$ iff $\exists w$ such that $(z, w) \in S$. □

Theorem

Let L be a Language, and $i \geq 1$. $L \in \Sigma_i P$ iff there is a polynomially balanced relation R such that the language $\{(x, y) : (x, y) \in R\}$ is in $\Pi_{i-1} P$ and $L = \{x$: there is a y such that $(x, y) \in R\}$.

Proof.

We will show this by using induction on i.
If $i = 1$ then this reduces to proposition 9.1.
If $i > 1$ then suppose such a relation R exists, to show that $L \in \Sigma_i$P we will construct machine M which guessed an appropriate y and asks a Σ_{i-1}P oracle whether $(x, y) \notin R$.
Conversely we can assume that $L \in \Sigma_i$P. By the definition of Σ_iP there is a NDTM M^K using oracle $K \in \Sigma_{i-1}$P. Thus, by induction, there is a relation S recognizable in Π_{i-2}P such that $z \in K$ iff $\exists w$ such that $(z, w) \in S$. □

cont.

Let $x \in L$ thus one computation of $M^K(x)$ halts on an accepting configuration. Thus we define R as follows, $(x, y) \in R$ iff y records an accepting computation of M^K on input x together with a certificate w_i for each z_i where z_i was a "yes" query to K and $(z_i, w_i) \in S$.

This can be done in Π_{i-1}P. The verification that each step of M^K is legal can be done in polynomial time. Each of the polynomially many "yes" queries can ,by induction, be done in Π_{i-2}P. And for each of the "no" queries we need to verify if $z_i \notin K$. But as $K \in \Sigma_{i-1}$P this can also be done in Π_{i-1}P. As each of these computations is in Π_{i-1}P the entire verification of $(x, y) \in R$ can be computed in Π_{i-1}P. □

cont.

Let $x \in L$ thus one computation of $M^K(x)$ halts on an accepting configuration. Thus we define R as follows, $(x, y) \in R$ iff y records an accepting computation of M^K on input x together with a certificate w_i for each z_i where z_i was a "yes" query to K and $(z_i, w_i) \in S$.

This can be done in Π_{i-1}P. The verification that each step of M^K is legal can be done in polynomial time. Each of the polynomially many "yes" queries can ,by induction, be done in Π_{i-2}P. And for each of the "no" queries we need to verify if $z_i \notin K$. But as $K \in \Sigma_{i-1}$P this can also be done in Π_{i-1}P. As each of these computations is in Π_{i-1}P the entire verification of $(x, y) \in R$ can be computed in Π_{i-1}P. □

cont.

Let $x \in L$ thus one computation of $M^K(x)$ halts on an accepting configuration. Thus we define R as follows, $(x, y) \in R$ iff y records an accepting computation of M^K on input x together with a certificate w_i for each z_i where z_i was a "yes" query to K and $(z_i, w_i) \in S$.

This can be done in Π_{i-1}P. The verification that each step of M^K is legal can be done in polynomial time. Each of the polynomially many "yes" queries can ,by induction, be done in Π_{i-2}P. And for each of the "no" queries we need to verify if $z_i \notin K$. But as $K \in \Sigma_{i-1}$P this can also be done in Π_{i-1}P. As each of these computations is in Π_{i-1}P the entire verification of $(x, y) \in R$ can be computed in Π_{i-1}P. □

cont.

Let $x \in L$ thus one computation of $M^K(x)$ halts on an accepting configuration. Thus we define R as follows, $(x, y) \in R$ iff y records an accepting computation of M^K on input x together with a certificate w_i for each z_i where z_i was a "yes" query to K and $(z_i, w_i) \in S$.
This can be done in Π_{i-1}P. The verification that each step of M^K is legal can be done in polynomial time. Each of the polynomially many "yes" queries can ,by induction, be done in Π_{i-2}P. And for each of the "no" queries we need to verify if $z_i \notin K$. But as $K \in \Sigma_{i-1}$P this can also be done in Π_{i-1}P. As each of these computations is in Π_{i-1}P the entire verification of $(x, y) \in R$ can be computed in Π_{i-1}P. □

cont.

Let $x \in L$ thus one computation of $M^K(x)$ halts on an accepting configuration. Thus we define R as follows, $(x, y) \in R$ iff y records an accepting computation of M^K on input x together with a certificate w_i for each z_i where z_i was a "yes" query to K and $(z_i, w_i) \in S$.
This can be done in Π_{i-1}P. The verification that each step of M^K is legal can be done in polynomial time. Each of the polynomially many "yes" queries can ,by induction, be done in Π_{i-2}P. And for each of the "no" queries we need to verify if $z_i \notin K$. But as $K \in \Sigma_{i-1}$P this can also be done in Π_{i-1}P. As each of these computations is in Π_{i-1}P the entire verification of $(x, y) \in R$ can be computed in Π_{i-1}P. □

cont.

Let $x \in L$ thus one computation of $M^K(x)$ halts on an accepting configuration. Thus we define R as follows, $(x, y) \in R$ iff y records an accepting computation of M^K on input x together with a certificate w_i for each z_i where z_i was a "yes" query to K and $(z_i, w_i) \in S$.

This can be done in Π_{i-1}P. The verification that each step of M^K is legal can be done in polynomial time. Each of the polynomially many "yes" queries can ,by induction, be done in Π_{i-2}P. And for each of the "no" queries we need to verify if $z_i \notin K$. But as $K \in \Sigma_{i-1}$P this can also be done in Π_{i-1}P. As each of these computations is in Π_{i-1}P the entire verification of $(x, y) \in R$ can be computed in Π_{i-1}P. □

Corollaries

Corollary

Let L be a Language, and $i \geq 1$. $L \in \Pi_i P$ iff there is a polynomially balanced relation R such that the language $\{(x,y) : (x,y) \in R\}$ is in $\Sigma_{i-1}P$ and $L = \{x$: for all y with $|y| < |x|^k$, $(x,y) \in R\}$.

Proof.

$\Pi_i P = \text{co}\Sigma_i P$. □

Corollary

Let L be a Language, and $i \geq 1$. $L \in \Sigma_i P$ iff there is a polynomially balanced polynomial-time decidable $(i+1)$-ary relation R such that $L = \{x: \exists y_1 \forall y_2 \exists y_3 \ldots Q y_i$ such that $(x, y_1, \ldots, y_i) \in R\}$ Where the ith quantifier is $\exists$ if i is odd $\forall$ otherwise.

Proof.

Starting with L, reduce each language in $\Pi_j P$ or $\Sigma_j P$ with its certificate form and do so for the resulting language of certificates. □

Corollaries

Corollary

Let L be a Language, and $i \geq 1$. $L \in \Pi_i P$ iff there is a polynomially balanced relation R such that the language $\{(x, y) : (x, y) \in R\}$ is in $\Sigma_{i-1}P$ and $L = \{x$: for all y with $|y| < |x|^k$, $(x, y) \in R\}$.

Proof.

Π_iP = coΣ_iP. □

Corollary

Let L be a Language, and $i \geq 1$. $L \in \Sigma_i P$ iff there is a polynomially balanced polynomial-time decidable $(i+1)$-ary relation R such that $L = \{x: \exists y_1 \forall y_2 \exists y_3 \ldots Q y_i$ such that $(x, y_1, \ldots, y_i) \in R\}$ Where the ith quantifier is $\exists$ if i is odd $\forall$ otherwise.

Proof.

Starting with *L*, reduce each language in Π_jP or Σ_jP with its certificate form and do so for the resulting language of certificates. □

Corollaries

Corollary

Let L be a Language, and $i \geq 1$. $L \in \Pi_i P$ iff there is a polynomially balanced relation R such that the language $\{(x, y) : (x, y) \in R\}$ is in $\Sigma_{i-1} P$ and $L = \{x$: for all y with $|y| < |x|^k, (x, y) \in R\}$.

Proof.

Π_iP = coΣ_iP. □

Corollary

Let L be a Language, and $i \geq 1$. $L \in \Sigma_i P$ iff there is a polynomially balanced polynomial-time decidable $(i + 1)$-ary relation R such that $L = \{x$: $\exists y_1 \forall y_2 \exists y_3 \ldots Q y_i$ such that $(x, y_1, \ldots, y_i) \in R\}$ Where the ith quantifier is $\exists$ if i is odd $\forall$ otherwise.

Proof.

Starting with L, reduce each language in Π_jP or Σ_jP with its certificate form and do so for the resulting language of certificates. □

Corollaries

Corollary

Let L be a Language, and $i \geq 1$. $L \in \Pi_i P$ iff there is a polynomially balanced relation R such that the language $\{(x, y) : (x, y) \in R\}$ is in $\Sigma_{i-1} P$ and $L = \{x$: for all y with $|y| < |x|^k$, $(x, y) \in R\}$.

Proof.

Π_iP = coΣ_iP. □

Corollary

Let L be a Language, and $i \geq 1$. $L \in \Sigma_i P$ iff there is a polynomially balanced polynomial-time decidable $(i + 1)$-ary relation R such that $L = \{x: \exists y_1 \forall y_2 \exists y_3 \ldots Q y_i$ such that $(x, y_1, \ldots, y_i) \in R\}$ Where the ith quantifier is $\exists$ if i is odd $\forall$ otherwise.

Proof.

Starting with L, reduce each language in Π_jP or Σ_jP with its certificate form and do so for the resulting language of certificates. □

Theorem

If for some $i \geq 1$, $\Sigma_i P = \Pi_i P$ then for all $j > i$ $\Sigma_j P = \Pi_j P = \Delta_j P = \Sigma_i P$.

Proof.

It suffices to show that Σ_iP = Π_iP implies Σ_iP = Σ_{i+1}P. Let $L \in \Sigma_{i+1}$P, by the previous theorem there is a relation R in Π_iP with $L = \{x$: there is a y such that $(x, y) \in R\}$. But by the assumption $R \in \Sigma_i$P. Thus there is a relation S in Π_{i-1}P with $R = \{(x, y)$: there is a z such that $(x, y, z) \in S\}$. Thus $L = \{x$: there is a (y, z) such that $(x, y, z) \in S\}$ meaning that $L \in \Sigma_i$P. □

Corollary

If P = NP or, NP = coNP, then the polynomial hierarchy collapses to the first level.

Theorem

If for some $i \geq 1$, $\Sigma_i P = \Pi_i P$ then for all $j > i$ $\Sigma_j P = \Pi_j P = \Delta_j P = \Sigma_i P$.

Proof.

It suffices to show that $\Sigma_i P = \Pi_i P$ implies $\Sigma_i P = \Sigma_{i+1} P$. Let $L \in \Sigma_{i+1} P$, by the previous theorem there is a relation R in $\Pi_i P$ with $L = \{x$: there is a y such that $(x, y) \in R\}$. But by the assumption $R \in \Sigma_i P$. Thus there is a relation S in $\Pi_{i-1} P$ with $R = \{(x, y)$: there is a z such that $(x, y, z) \in S\}$. Thus $L = \{x$: there is a (y, z) such that $(x, y, z) \in S\}$ meaning that $L \in \Sigma_i P$. □

Corollary

If P = NP or, NP = coNP, then the polynomial hierarchy collapses to the first level.

Theorem

If for some $i \geq 1$, $\Sigma_i P = \Pi_i P$ then for all $j > i$ $\Sigma_j P = \Pi_j P = \Delta_j P = \Sigma_i P$.

Proof.

It suffices to show that $\Sigma_i P = \Pi_i P$ implies $\Sigma_i P = \Sigma_{i+1} P$. Let $L \in \Sigma_{i+1} P$, by the previous theorem there is a relation R in $\Pi_i P$ with $L = \{x\colon$ there is a y such that $(x, y) \in R\}$. But by the assumption $R \in \Sigma_i P$. Thus there is a relation S in $\Pi_{i-1} P$ with $R = \{(x, y)\colon$ there is a z such that $(x, y, z) \in S\}$. Thus $L = \{x\colon$ there is a (y, z) such that $(x, y, z) \in S\}$ meaning that $L \in \Sigma_i P$. □

Corollary

If P = NP or, NP = coNP, then the polynomial hierarchy collapses to the first level.

Theorem

If for some $i \geq 1$, $\Sigma_i P = \Pi_i P$ then for all $j > i$ $\Sigma_j P = \Pi_j P = \Delta_j P = \Sigma_i P$.

Proof.

It suffices to show that Σ_iP = Π_iP implies Σ_iP = Σ_{i+1}P. Let $L \in \Sigma_{i+1}$P, by the previous theorem there is a relation R in Π_iP with $L = \{x$: there is a y such that $(x, y) \in R\}$. But by the assumption $R \in \Sigma_i$P. Thus there is a relation S in Π_{i-1}P with $R = \{(x, y)$: there is a z such that $(x, y, z) \in S\}$. Thus $L = \{x$: there is a (y, z) such that $(x, y, z) \in S\}$ meaning that $L \in \Sigma_i$P. □

Corollary

If P = NP or, NP = coNP, then the polynomial hierarchy collapses to the first level.

Theorem

If for some $i \geq 1$, $\Sigma_i P = \Pi_i P$ then for all $j > i$ $\Sigma_j P = \Pi_j P = \Delta_j P = \Sigma_i P$.

Proof.

It suffices to show that Σ_iP = Π_iP implies Σ_iP = Σ_{i+1}P. Let $L \in \Sigma_{i+1}$P, by the previous theorem there is a relation R in Π_iP with $L = \{x\colon$ there is a y such that $(x, y) \in R\}$. But by the assumption $R \in \Sigma_i$P. Thus there is a relation S in Π_{i-1}P with $R = \{(x, y)\colon$ there is a z such that $(x, y, z) \in S\}$. Thus $L = \{x\colon$ there is a (y, z) such that $(x, y, z) \in S\}$ meaning that $L \in \Sigma_i$P. □

Corollary

If P = NP or, NP = coNP, then the polynomial hierarchy collapses to the first level.

Theorem

If for some $i \geq 1$, $\Sigma_i P = \Pi_i P$ then for all $j > i$ $\Sigma_j P = \Pi_j P = \Delta_j P = \Sigma_i P$.

Proof.

It suffices to show that Σ_iP = Π_iP implies Σ_iP = Σ_{i+1}P. Let $L \in \Sigma_{i+1}$P, by the previous theorem there is a relation R in Π_iP with $L = \{x\colon$ there is a y such that $(x, y) \in R\}$. But by the assumption $R \in \Sigma_i$P. Thus there is a relation S in Π_{i-1}P with $R = \{(x, y)\colon$ there is a z such that $(x, y, z) \in S\}$. Thus $L = \{x\colon$ there is a (y, z) such that $(x, y, z) \in S\}$ meaning that $L \in \Sigma_i$P. □

Corollary

If P = NP or, NP = coNP, then the polynomial hierarchy collapses to the first level.

Definition ($QSAT_i$)

Let ϕ be a boolean expression with its boolean variables partitioned into i sets $X_1, X_2, \ldots, X_i$ we have that the expression $\exists X_1 \forall X_2 \exists X_3 \ldots QX_i \phi$ where the quantifies alternate is in $QSAT_i$.

Theorem

$QSAT_i$ is Σ_iP-complete.

Proof.

By the second corollary $QSAT_i \in \Sigma_i$P.
Let $L \in \Sigma_i$P. We convert L to the form from the second corollary. Since the resulting relation R can be decided in polynomial time there is a polynomial time Turing Machine M which accepts precisely the strings $x; y_1; \ldots; y_i$ such that$(x, y_1, \ldots, y_i) \in R$.
Suppose that i is odd. By Cook's theorem we can write a boolean formula ϕ which captures the computation of M. We can divide the variables of ϕ into $i + 2$ groups $X, Y_1, \ldots, Y_i$,the *input variables*, which contain the variables representing the symbols in the $x, y_1, \ldots, y_i$ substrings of the input. And a group Z which incorporates the remaining variables. □

Definition ($QSAT_i$)

Let ϕ be a boolean expression with its boolean variables partitioned into i sets $X_1, X_2, \ldots, X_i$ we have that the expression $\exists X_1 \forall X_2 \exists X_3 \ldots QX_i \phi$ where the quantifies alternate is in $QSAT_i$.

Theorem

$QSAT_i$ is Σ_iP-complete.

Proof.

By the second corollary $QSAT_i \in \Sigma_i$P.
Let $L \in \Sigma_i$P. We convert L to the form from the second corollary. Since the resulting relation R can be decided in polynomial time there is a polynomial time Turing Machine M which accepts precisely the strings $x; y_1; \ldots; y_i$ such that$(x, y_1, \ldots, y_i) \in R$. Suppose that i is odd. By Cook's theorem we can write a boolean formula ϕ which captures the computation of M. We can divide the variables of ϕ into $i + 2$ groups $X, Y_1, \ldots, Y_i$,the *input variables*, which contain the variables representing the symbols in the $x, y_1, \ldots, y_i$ substrings of the input. And a group Z which incorporates the remaining variables. □

Definition ($QSAT_i$)

Let ϕ be a boolean expression with its boolean variables partitioned into i sets $X_1, X_2, \ldots, X_i$ we have that the expression $\exists X_1 \forall X_2 \exists X_3 \ldots QX_i\phi$ where the quantifies alternate is in $QSAT_i$.

Theorem

$QSAT_i$ is Σ_iP-complete.

Proof.

By the second corollary $QSAT_i \in \Sigma_i$P.
Let $L \in \Sigma_i$P. We convert L to the form from the second corollary. Since the resulting relation R can be decided in polynomial time there is a polynomial time Turing Machine M which accepts precisely the strings $x; y_1; \ldots; y_i$ such that$(x, y_1, \ldots, y_i) \in R$. Suppose that i is odd. By Cook's theorem we can write a boolean formula ϕ which captures the computation of M. We can divide the variables of ϕ into $i + 2$ groups $X, Y_1, \ldots, Y_i$,the *input variables*, which contain the variables representing the symbols in the $x, y_1, \ldots, y_i$ substrings of the input. And a group Z which incorporates the remaining variables. □

Definition ($QSAT_i$)

Let ϕ be a boolean expression with its boolean variables partitioned into i sets $X_1, X_2, \ldots, X_i$ we have that the expression $\exists X_1 \forall X_2 \exists X_3 \ldots QX_i \phi$ where the quantifies alternate is in $QSAT_i$.

Theorem

$QSAT_i$ is Σ_iP-complete.

Proof.

By the second corollary $QSAT_i \in \Sigma_i$P.
Let $L \in \Sigma_i$P. We convert L to the form from the second corollary. Since the resulting relation R can be decided in polynomial time there is a polynomial time Turing Machine M which accepts precisely the strings $x; y_1; \ldots; y_i$ such that$(x, y_1, \ldots, y_i) \in R$.
Suppose that i is odd. By Cook's theorem we can write a boolean formula ϕ which captures the computation of M. We can divide the variables of ϕ into $i + 2$ groups $X, Y_1, \ldots, Y_i$,the *input variables*, which contain the variables representing the symbols in the $x, y_1, \ldots, y_i$ substrings of the input. And a group Z which incorporates the remaining variables. □

Definition ($QSAT_i$)

Let ϕ be a boolean expression with its boolean variables partitioned into i sets $X_1, X_2, \ldots, X_i$ we have that the expression $\exists X_1 \forall X_2 \exists X_3 \ldots QX_i \phi$ where the quantifies alternate is in $QSAT_i$.

Theorem

$QSAT_i$ is Σ_iP-complete.

Proof.

By the second corollary $QSAT_i \in \Sigma_i$P.
Let $L \in \Sigma_i$P. We convert L to the form from the second corollary. Since the resulting relation R can be decided in polynomial time there is a polynomial time Turing Machine M which accepts precisely the strings $x; y_1; \ldots; y_i$ such that$(x, y_1, \ldots, y_i) \in R$.
Suppose that i is odd. By Cook's theorem we can write a boolean formula ϕ which captures the computation of M. We can divide the variables of ϕ into $i + 2$ groups $X, Y_1, \ldots, Y_i$,the *input variables*, which contain the variables representing the symbols in the $x, y_1, \ldots, y_i$ substrings of the input. And a group Z which incorporates the remaining variables. □

Definition ($QSAT_i$)

Let ϕ be a boolean expression with its boolean variables partitioned into i sets $X_1, X_2, \ldots, X_i$ we have that the expression $\exists X_1 \forall X_2 \exists X_3 \ldots QX_i \phi$ where the quantifies alternate is in $QSAT_i$.

Theorem

$QSAT_i$ is $\Sigma_i P$-complete.

Proof.

By the second corollary $QSAT_i \in \Sigma_i$P.
Let $L \in \Sigma_i$P. We convert L to the form from the second corollary. Since the resulting relation R can be decided in polynomial time there is a polynomial time Turing Machine M which accepts precisely the strings $x; y_1; \ldots; y_i$ such that$(x, y_1, \ldots, y_i) \in R$.
Suppose that i is odd. By Cook's theorem we can write a boolean formula ϕ which captures the computation of M. We can divide the variables of ϕ into $i + 2$ groups $X, Y_1, \ldots, Y_i$,the *input variables*, which contain the variables representing the symbols in the $x, y_1, \ldots, y_i$ substrings of the input. And a group Z which incorporates the remaining variables. □

cont.

If the variables in $X, Y_1, \ldots, Y_i$ are fixed then the resulting expression is satisfiable iff the input variables spell out a string accepted by M. Let x' be any string, and substitute into ϕ the corresponding boolean variables X' for X. We know that $x' \in L$ iff there is a y_1, such that for all $y_2, \ldots,$ there is a y_i such that $(x', y_1, y_2, \ldots, y_i) \in R$ however this is equivalent to stating that $\exists Y_1 \forall Y_2 \ldots \exists Y_i; Z \phi(X')$. A similar proof holds if i is odd. □

cont.

If the variables in $X, Y_1, \ldots, Y_i$ are fixed then the resulting expression is satisfiable iff the input variables spell out a string accepted by M. Let x' be any string, and substitute into ϕ the corresponding boolean variables X' for X. We know that $x' \in L$ iff there is a y_1, such that for all $y_2, \ldots,$ there is a y_i such that $(x', y_1, y_2, \ldots, y_i) \in R$ however this is equivalent to stating that $\exists Y_1 \forall Y_2 \ldots \exists Y_i; Z\, \phi(X')$. A similar proof holds if i is odd. □

cont.

If the variables in $X, Y_1, \ldots, Y_i$ are fixed then the resulting expression is satisfiable iff the input variables spell out a string accepted by M. Let x' be any string, and substitute into ϕ the corresponding boolean variables X' for X. We know that $x' \in L$ iff there is a y_1, such that for all $y_2, \ldots,$ there is a y_i such that $(x', y_1, y_2, \ldots, y_i) \in R$ however this is equivalent to stating that $\exists Y_1 \forall Y_2 \ldots \exists Y_i; Z\,\phi(X')$. A similar proof holds if i is odd. □

cont.

If the variables in $X, Y_1, \ldots, Y_i$ are fixed then the resulting expression is satisfiable iff the input variables spell out a string accepted by M. Let x' be any string, and substitute into ϕ the corresponding boolean variables X' for X. We know that $x' \in L$ iff there is a y_1, such that for all $y_2, \ldots$, there is a y_i such that $(x', y_1, y_2, \ldots, y_i) \in R$ however this is equivalent to stating that $\exists Y_1 \forall Y_2 \ldots \exists Y_i; Z \phi(X')$. A similar proof holds if i is odd. □

cont.

If the variables in $X, Y_1, \ldots, Y_i$ are fixed then the resulting expression is satisfiable iff the input variables spell out a string accepted by M. Let x' be any string, and substitute into ϕ the corresponding boolean variables X' for X. We know that $x' \in L$ iff there is a y_1, such that for all $y_2, \ldots,$ there is a y_i such that $(x', y_1, y_2, \ldots, y_i) \in R$ however this is equivalent to stating that $\exists Y_1 \forall Y_2 \ldots \exists Y_i; Z \phi(X')$. A similar proof holds if i is odd. □

Theorem

If there is a **PH-complete** *problem then the polynomial hierarchy collapses to some finite level.*

Proof.

Suppose that L is **PH-complete**. Since $L \in$ **PH**, there is an $i \geq 0$ for which $L \in \Sigma_i$P. However any language L' in Σ_{i+1}P reduces to L. since the levels of the polynomial hierarchy are closed under reductions $L' \in \Sigma_i$P and so Σ_iP = Σ_{i+1}P. □

proposition

PH $\subseteq$ PSPACE.

Theorem

If there is a **PH-complete** *problem then the polynomial hierarchy collapses to some finite level.*

Proof.

Suppose that L is **PH-complete**. Since $L \in$ **PH**, there is an $i \geq 0$ for which $L \in \Sigma_i$P. However any language L' in Σ_{i+1}P reduces to L. since the levels of the polynomial hierarchy are closed under reductions $L' \in \Sigma_i$P and so Σ_iP = Σ_{i+1}P. □

proposition

PH $\subseteq$ PSPACE.

Theorem

If there is a **PH-complete** *problem then the polynomial hierarchy collapses to some finite level.*

Proof.

Suppose that L is **PH-complete**. Since $L \in$ **PH**, there is an $i \geq 0$ for which $L \in \Sigma_i$P. However any language L' in Σ_{i+1}P reduces to L. since the levels of the polynomial hierarchy are closed under reductions $L' \in \Sigma_i$P and so Σ_iP = Σ_{i+1}P. □

proposition

PH $\subseteq$ PSPACE.

Theorem

If there is a **PH-complete** *problem then the polynomial hierarchy collapses to some finite level.*

Proof.

Suppose that L is **PH-complete**. Since $L \in$ **PH**, there is an $i \geq 0$ for which $L \in \Sigma_i$P. However any language L' in Σ_{i+1}P reduces to L. since the levels of the polynomial hierarchy are closed under reductions $L' \in \Sigma_i$P and so Σ_iP = Σ_{i+1}P. □

proposition

PH $\subseteq$ PSPACE.

Theorem

If there is a **PH-complete** *problem then the polynomial hierarchy collapses to some finite level.*

Proof.

Suppose that L is **PH-complete**. Since $L \in$ **PH**, there is an $i \geq 0$ for which $L \in \Sigma_i$P. However any language L' in Σ_{i+1}P reduces to L. since the levels of the polynomial hierarchy are closed under reductions $L' \in \Sigma_i$P and so Σ_iP = Σ_{i+1}P. □

proposition

PH $\subseteq$ PSPACE.

Theorem

If there is a **PH-complete** *problem then the polynomial hierarchy collapses to some finite level.*

Proof.

Suppose that L is **PH-complete**. Since $L \in$ **PH**, there is an $i \geq 0$ for which $L \in \Sigma_i$P. However any language L' in Σ_{i+1}P reduces to L. since the levels of the polynomial hierarchy are closed under reductions $L' \in \Sigma_i$P and so Σ_iP = Σ_{i+1}P. □

proposition

PH $\subseteq$ PSPACE.

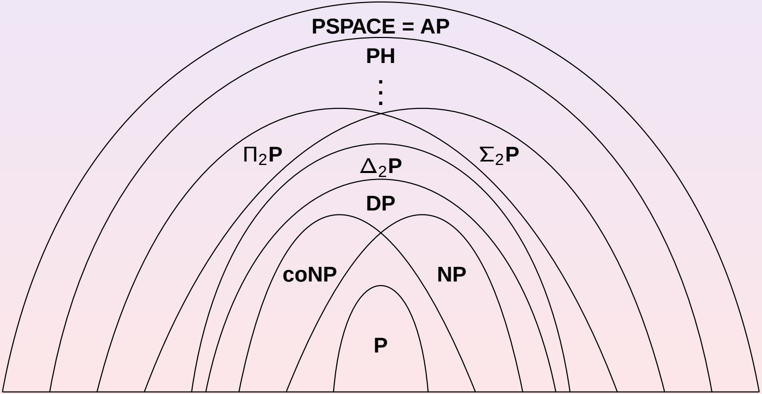
Other problems in **PH**

Examples

1. MINIMUM EQUIVALENT CIRCUIT $\in \Sigma_2$P
   Given a boolean circuit C and integer k is there a boolean circuit C' of size less than or equal to k such that for all possible inputs $C = C'$.

Outline

1. The complexity class DP
   - Definition of DP
   - Problems in DP
2. The classes P^{NP} and FP^{NP}
   - The definition of P^{NP} and FP^{NP}
3. The polynomial Hierarchy
   - The definition of the Polynomial Hierarchy
   - Examining the Polynomial Hierarchy
   - Diagram of the complexity classes
4. A look a PSPACE
   - QSAT is PSPACE complete
   - PSPACE=AP
   - Geography is PSPACE-complete


PSPACE = AP
PH
⋮
Π₂P
Σ₂P
Δ₂P
DP
coNP
NP
P

Outline

Definition (QSAT)

Let ϕ be a boolean expression with n variables then the expression $\exists x_1 \forall x_2 \ldots Q_n x_n$ where the quantifiers alternate is a QSAT expression.

Theorem

QSAT is PSPACE complete

Proof.

Part 1, QSAT $\in$ PSPACE
The QSAT expression be converted into a boolean circuit as follows. We construct a full binary tree with the ith level branching to represent the possible assignments for x_i and the leaves representing the result os substituting the corresponding assignment int ϕ. The interior nodes are then converted to and gates at even levels and or gates at odd levels. The resulting circuit can be evaluated in $O(n)$ space. The entire circuit cannot be stored as it is exponential in size, however space bounded algorithms can be combined. □

Definition (QSAT)

Let ϕ be a boolean expression with n variables then the expression $\exists x_1 \forall x_2 \ldots Q_n x_n$ where the quantifiers alternate is a QSAT expression.

Theorem

QSAT is PSPACE complete

Proof.

Part 1, QSAT $\in$ PSPACE

The QSAT expression be converted into a boolean circuit as follows. We construct a full binary tree with the ith level branching to represent the possible assignments for x_i and the leaves representing the result os substituting the corresponding assignment int ϕ. The interior nodes are then converted to and gates at even levels and or gates at odd levels. The resulting circuit can be evaluated in $O(n)$ space. The entire circuit cannot be stored as it is exponential in size, however space bounded algorithms can be combined. □

Definition (QSAT)

Let ϕ be a boolean expression with n variables then the expression $\exists x_1 \forall x_2 \ldots Q_n x_n$ where the quantifiers alternate is a QSAT expression.

Theorem

QSAT is PSPACE complete

Proof.

Part 1, QSAT $\in$ PSPACE

The QSAT expression be converted into a boolean circuit as follows. We construct a full binary tree with the ith level branching to represent the possible assignments for x_i and the leaves representing the result os substituting the corresponding assignment int ϕ. The interior nodes are then converted to and gates at even levels and or gates at odd levels. The resulting circuit can be evaluated in $O(n)$ space. The entire circuit cannot be stored as it is exponential in size, however space bounded algorithms can be combined. □

Definition (QSAT)

Let ϕ be a boolean expression with n variables then the expression $\exists x_1 \forall x_2 \ldots Q_n x_n$ where the quantifiers alternate is a QSAT expression.

Theorem

QSAT is PSPACE complete

Proof.

Part 1, QSAT $\in$ PSPACE
The QSAT expression be converted into a boolean circuit as follows. We construct a full binary tree with the ith level branching to represent the possible assignments for x_i and the leaves representing the result os substituting the corresponding assignment int ϕ. The interior nodes are then converted to and gates at even levels and or gates at odd levels. The resulting circuit can be evaluated in $O(n)$ space. The entire circuit cannot be stored as it is exponential in size, however space bounded algorithms can be combined. □

Definition (QSAT)

Let ϕ be a boolean expression with n variables then the expression $\exists x_1 \forall x_2 \ldots Q_n x_n$ where the quantifiers alternate is a QSAT expression.

Theorem

QSAT is PSPACE complete

Proof.

Part 1, QSAT $\in$ PSPACE
The QSAT expression be converted into a boolean circuit as follows. We construct a full binary tree with the ith level branching to represent the possible assignments for x_i and the leaves representing the result os substituting the corresponding assignment int ϕ. The interior nodes are then converted to and gates at even levels and or gates at odd levels. The resulting circuit can be evaluated in $O(n)$ space. The entire circuit cannot be stored as it is exponential in size, however space bounded algorithms can be combined. □

Definition (QSAT)

Let ϕ be a boolean expression with n variables then the expression $\exists x_1 \forall x_2 \ldots Q_n x_n$ where the quantifiers alternate is a QSAT expression.

Theorem

QSAT is PSPACE complete

Proof.

Part 1, QSAT $\in$ PSPACE
The QSAT expression be converted into a boolean circuit as follows. We construct a full binary tree with the ith level branching to represent the possible assignments for x_i and the leaves representing the result os substituting the corresponding assignment int ϕ. The interior nodes are then converted to and gates at even levels and or gates at odd levels. The resulting circuit can be evaluated in $O(n)$ space. The entire circuit cannot be stored as it is exponential in size, however space bounded algorithms can be combined. □

Definition (QSAT)

Let ϕ be a boolean expression with n variables then the expression $\exists x_1 \forall x_2 \ldots Q_n x_n$ where the quantifiers alternate is a QSAT expression.

Theorem

QSAT is PSPACE complete

Proof.

Part 1, QSAT $\in$ PSPACE
The QSAT expression be converted into a boolean circuit as follows. We construct a full binary tree with the ith level branching to represent the possible assignments for x_i and the leaves representing the result os substituting the corresponding assignment int ϕ. The interior nodes are then converted to and gates at even levels and or gates at odd levels. The resulting circuit can be evaluated in $O(n)$ space. The entire circuit cannot be stored as it is exponential in size, however space bounded algorithms can be combined. □

cont.

The proof of this relies on reachability and essentially a restatement of Savitch's Theorem. Suppose that M decides L in polynomial space. Let xL be an input to M of size n and consider the configuration graph of M on input x. We know that for some integer k it has at most 2^{n^k} configurations. So each configuration can be encoded as a n^k bit vector. For each integer i we will now compute a boolean expression ψ_i with free variables in the set $A \cup B = \{a_1, \dots, a_{n^k}, b_1, \dots, b_{n^k}\}$ such that ψ_i is true iff for a truth assignment that corresponds to two states $a = a_1 \dots a_{n^k}$ and $b = b_1 \dots b_{n^k}$ if there is a path between a and b in the configuration graph of length at most 2^i.
$\psi_0(a, b)$ simply states that the configurations a and b are equal or that a follows from b in one step. ψ_0 can be written as the disjunction of $O(n^k)$ implicants each containing $O(n^k)$ literals. When computing ψ_{i+1} from ψ_i setting $\psi_{i+1} = \exists z[\psi_i(a, z) \wedge \psi_i(z, b)]$ is unfeasible as it produces exponentially large expressions. □

cont.

The proof of this relies on reachability and essentially a restatement of Savitch's Theorem. Suppose that M decides L in polynomial space. Let xL be an input to M of size n and consider the configuration graph of M on input x. We know that for some integer k it has at most 2^{n^k} configurations. So each configuration can be encoded as a n^k bit vector. For each integer i we will now compute a boolean expression ψ_i with free variables in the set $A \cup B = \{a_1, \ldots, a_{n^k}, b_1, \ldots, b_{n^k}\}$ such that ψ_i is true iff for a truth assignment that corresponds to two states $a = a_1 \ldots a_{n^k}$ and $b = b_1 \ldots b_{n^k}$ if there is a path between a and b in the configuration graph of length at most 2^i.

$\psi_0(a, b)$ simply states that the configurations a and b are equal or that a follows from b in one step. ψ_0 can be written as the disjunction of $O(n^k)$ implicants each containing $O(n^k)$ literals. When computing ψ_{i+1} from ψ_i setting $\psi_{i+1} = \exists z[\psi_i(a, z) \wedge \psi_i(z, b)]$ is unfeasible as it produces exponentially large expressions. □

cont.

The proof of this relies on reachability and essentially a restatement of Savitch's Theorem. Suppose that M decides L in polynomial space. Let xL be an input to M of size n and consider the configuration graph of M on input x. We know that for some integer k it has at most 2^{n^k} configurations. So each configuration can be encoded as a n^k bit vector. For each integer i we will now compute a boolean expression ψ_i with free variables in the set $A \cup B = \{a_1, \ldots, a_{n^k}, b_1, \ldots, b_{n^k}\}$ such that ψ_i is true iff for a truth assignment that corresponds to two states $a = a_1 \ldots a_{n^k}$ and $b = b_1 \ldots b_{n^k}$ if there is a path between a and b in the configuration graph of length at most 2^i. $\psi_0(a, b)$ simply states that the configurations a and b are equal or that a follows from b in one step. ψ_0 can be written as the disjunction of $O(n^k)$ implicants each containing $O(n^k)$ literals. When computing ψ_{i+1} from ψ_i setting $\psi_{i+1} = \exists z[\psi_i(a, z) \wedge \psi_i(z, b)]$ is unfeasible as it produces exponentially large expressions. □

cont.

The proof of this relies on reachability and essentially a restatement of Savitch's Theorem. Suppose that M decides L in polynomial space. Let xL be an input to M of size n and consider the configuration graph of M on input x. We know that for some integer k it has at most 2^{n^k} configurations. So each configuration can be encoded as a n^k bit vector. For each integer i we will now compute a boolean expression ψ_i with free variables in the set $A \cup B = \{a_1, \ldots, a_{n^k}, b_1, \ldots, b_{n^k}\}$ such that ψ_i is true iff for a truth assignment that corresponds to two states $a = a_1 \ldots a_{n^k}$ and $b = b_1 \ldots b_{n^k}$ if there is a path between a and b in the configuration graph of length at most 2^i.

$\psi_0(a, b)$ simply states that the configurations a and b are equal or that a follows from b in one step. ψ_0 can be written as the disjunction of $O(n^k)$ implicants each containing $O(n^k)$ literals. When computing ψ_{i+1} from ψ_i setting $\psi_{i+1} = \exists z[\psi_i(a, z) \wedge \psi_i(z, b)]$ is unfeasible as it produces exponentially large expressions. □

cont.

The proof of this relies on reachability and essentially a restatement of Savitch's Theorem. Suppose that M decides L in polynomial space. Let xL be an input to M of size n and consider the configuration graph of M on input x. We know that for some integer k it has at most 2^{n^k} configurations. So each configuration can be encoded as a n^k bit vector. For each integer i we will now compute a boolean expression ψ_i with free variables in the set $A \cup B = \{a_1, \ldots, a_{n^k}, b_1, \ldots, b_{n^k}\}$ such that ψ_i is true iff for a truth assignment that corresponds to two states $a = a_1 \ldots a_{n^k}$ and $b = b_1 \ldots b_{n^k}$ if there is a path between a and b in the configuration graph of length at most 2^i.
$\psi_0(a, b)$ simply states that the configurations a and b are equal or that a follows from b in one step. ψ_0 can be written as the disjunction of $O(n^k)$ implicants each containing $O(n^k)$ literals. When computing ψ_{i+1} from ψ_i setting $\psi_{i+1} = \exists z[\psi_i(a, z) \wedge \psi_i(z, b)]$ is unfeasible as it produces exponentially large expressions. □

cont.

The proof of this relies on reachability and essentially a restatement of Savitch's Theorem. Suppose that M decides L in polynomial space. Let xL be an input to M of size n and consider the configuration graph of M on input x. We know that for some integer k it has at most 2^{n^k} configurations. So each configuration can be encoded as a n^k bit vector. For each integer i we will now compute a boolean expression ψ_i with free variables in the set $A \cup B = \{a_1, \ldots, a_{n^k}, b_1, \ldots, b_{n^k}\}$ such that ψ_i is true iff for a truth assignment that corresponds to two states $a = a_1 \ldots a_{n^k}$ and $b = b_1 \ldots b_{n^k}$ if there is a path between a and b in the configuration graph of length at most 2^i.

$\psi_0(a, b)$ simply states that the configurations a and b are equal or that a follows from b in one step. ψ_0 can be written as the disjunction of $O(n^k)$ implicants each containing $O(n^k)$ literals. When computing ψ_{i+1} from ψ_i setting $\psi_{i+1} = \exists z[\psi_i(a, z) \wedge \psi_i(z, b)]$ is unfeasible as it produces exponentially large expressions. □

cont.

The proof of this relies on reachability and essentially a restatement of Savitch's Theorem. Suppose that M decides L in polynomial space. Let xL be an input to M of size n and consider the configuration graph of M on input x. We know that for some integer k it has at most 2^{n^k} configurations. So each configuration can be encoded as a n^k bit vector. For each integer i we will now compute a boolean expression ψ_i with free variables in the set $A \cup B = \{a_1, \ldots, a_{n^k}, b_1, \ldots, b_{n^k}\}$ such that ψ_i is true iff for a truth assignment that corresponds to two states $a = a_1 \ldots a_{n^k}$ and $b = b_1 \ldots b_{n^k}$ if there is a path between a and b in the configuration graph of length at most 2^i.

$\psi_0(a, b)$ simply states that the configurations a and b are equal or that a follows from b in one step. ψ_0 can be written as the disjunction of $O(n^k)$ implicants each containing $O(n^k)$ literals. When computing ψ_{i+1} from ψ_i setting $\psi_{i+1} = \exists z[\psi_i(a, z) \wedge \psi_i(z, b)]$ is unfeasible as it produces exponentially large expressions. □

cont.

Instead we want to use only one copy of ψ_i so instead we have this expression $\psi_{i+1} = \exists z \forall x \forall y[((x = a \wedge y = z) \vee (x = z \wedge y = b)) \rightarrow \psi_i(a, y)]$ where x, y, z are blocks of n^k variables. Now that we have a valid definition of each ϕ_i we now need to get them into a from recognizable by QSAT. First we need to get it into prenex normal form however the quantifiers of ψ_i can simply be migrated to the front behind those of ψ_{i+1}. However the expression still needs to be converted to CNF form, however the space requirements are large so instead a conversion to DNF will be made. This can be done using the DNF of ψ_i followed by $16n^{2k}$ implicants. For integers $1 \leq i, j \leq n^k$ there are 16 implicants of the form $(x_i \wedge \overline{a_i} \wedge x_j \wedge \overline{z_j})$ each corresponding to a way to make $((x = a \wedge y = z) \vee (x = z \wedge y = b))$ false. To obtain ψ_{n^k} we add to ψ_0, n^k sets of implicants and prefix the result with n^k layers of quantifiers. Thus any problem in PSPACE can be reduced to coQSAT and so ant problem in coPSPACE = PSPACE can be reduced to QSAT. □

cont.

Instead we want to use only one copy of ψ_i so instead we have this expression $\psi_{i+1} = \exists z \forall x \forall y[((x = a \wedge y = z) \vee (x = z \wedge y = b)) \rightarrow \psi_i(a, y)]$ where x, y, z are blocks of n^k variables. Now that we have a valid definition of each ϕ_i we now need to get them into a from recognizable by QSAT. First we need to get it into prenex normal form however the quantifiers of ψ_i can simply be migrated to the front behind those of ψ_{i+1}. However the expression still needs to be converted to CNF form, however the space requirements are large so instead a conversion to DNF will be made. This can be done using the DNF of ψ_i followed by $16n^{2k}$ implicants. For integers $1 \leq i, j \leq n^k$ there are 16 implicants of the form $(x_i \wedge \overline{a_i} \wedge x_j \wedge \overline{z_j})$ each corresponding to a way to make $((x = a \wedge y = z) \vee (x = z \wedge y = b))$ false. To obtain ψ_{n^k} we add to ψ_0, n^k sets of implicants and prefix the result with n^k layers of quantifiers. Thus any problem in PSPACE can be reduced to coQSAT and so ant problem in coPSPACE = PSPACE can be reduced to QSAT. □

cont.

Instead we want to use only one copy of ψ_i so instead we have this expression $\psi_{i+1} = \exists z \forall x \forall y [((x = a \wedge y = z) \vee (x = z \wedge y = b)) \rightarrow \psi_i(a, y)]$ where x, y, z are blocks of n^k variables. Now that we have a valid definition of each ϕ_i we now need to get them into a from recognizable by QSAT. First we need to get it into prenex normal form however the quantifiers of ψ_i can simply be migrated to the front behind those of ψ_{i+1}. However the expression still needs to be converted to CNF form, however the space requirements are large so instead a conversion to DNF will be made. This can be done using the DNF of ψ_i followed by $16n^{2k}$ implicants. For integers $1 \leq i, j \leq n^k$ there are 16 implicants of the form $(x_i \wedge \overline{a_i} \wedge x_j \wedge \overline{z_j})$ each corresponding to a way to make $((x = a \wedge y = z) \vee (x = z \wedge y = b))$ false. To obtain ψ_{n^k} we add to ψ_0, n^k sets of implicants and prefix the result with n^k layers of quantifiers. Thus any problem in PSPACE can be reduced to coQSAT and so ant problem in coPSPACE = PSPACE can be reduced to QSAT. □

cont.

Instead we want to use only one copy of ψ_i so instead we have this expression $\psi_{i+1} = \exists z \forall x \forall y[((x = a \wedge y = z) \vee (x = z \wedge y = b)) \rightarrow \psi_i(a, y)]$ where x, y, z are blocks of n^k variables. Now that we have a valid definition of each ϕ_i we now need to get them into a from recognizable by QSAT. First we need to get it into prenex normal form however the quantifiers of ψ_i can simply be migrated to the front behind those of ψ_{i+1}. However the expression still needs to be converted to CNF form, however the space requirements are large so instead a conversion to DNF will be made. This can be done using the DNF of ψ_i followed by $16n^{2k}$ implicants. For integers $1 \leq i, j \leq n^k$ there are 16 implicants of the form $(x_i \wedge \overline{a_i} \wedge x_j \wedge \overline{z_j})$ each corresponding to a way to make $((x = a \wedge y = z) \vee (x = z \wedge y = b))$ false. To obtain ψ_{n^k} we add to ψ_0, n^k sets of implicants and prefix the result with n^k layers of quantifiers. Thus any problem in PSPACE can be reduced to coQSAT and so ant problem in coPSPACE = PSPACE can be reduced to QSAT. □

cont.

Instead we want to use only one copy of ψ_i so instead we have this expression $\psi_{i+1} = \exists z \forall x \forall y[((x = a \wedge y = z) \vee (x = z \wedge y = b)) \rightarrow \psi_i(a, y)]$ where x, y, z are blocks of n^k variables. Now that we have a valid definition of each ϕ_i we now need to get them into a from recognizable by QSAT. First we need to get it into prenex normal form however the quantifiers of ψ_i can simply be migrated to the front behind those of ψ_{i+1}. However the expression still needs to be converted to CNF form, however the space requirements are large so instead a conversion to DNF will be made. This can be done using the DNF of ψ_i followed by $16n^{2k}$ implicants. For integers $1 \leq i, j \leq n^k$ there are 16 implicants of the form $(x_i \wedge \overline{a_i} \wedge x_j \wedge \overline{z_j})$ each corresponding to a way to make $((x = a \wedge y = z) \vee (x = z \wedge y = b))$ false. To obtain ψ_{n^k} we add to ψ_0, n^k sets of implicants and prefix the result with n^k layers of quantifiers. Thus any problem in PSPACE can be reduced to coQSAT and so ant problem in coPSPACE = PSPACE can be reduced to QSAT. □

cont.

Instead we want to use only one copy of ψ_i so instead we have this expression $\psi_{i+1} = \exists z \forall x \forall y[((x = a \wedge y = z) \vee (x = z \wedge y = b)) \rightarrow \psi_i(a, y)]$ where x, y, z are blocks of n^k variables. Now that we have a valid definition of each ϕ_i we now need to get them into a from recognizable by QSAT. First we need to get it into prenex normal form however the quantifiers of ψ_i can simply be migrated to the front behind those of ψ_{i+1}. However the expression still needs to be converted to CNF form, however the space requirements are large so instead a conversion to DNF will be made. This can be done using the DNF of ψ_i followed by $16n^{2k}$ implicants. For integers $1 \leq i, j \leq n^k$ there are 16 implicants of the form $(x_i \wedge \overline{a_i} \wedge x_j \wedge \overline{z_j})$ each corresponding to a way to make $((x = a \wedge y = z) \vee (x = z \wedge y = b))$ false. To obtain ψ_{n^k} we add to ψ_0, n^k sets of implicants and prefix the result with n^k layers of quantifiers. Thus any problem in PSPACE can be reduced to coQSAT and so ant problem in coPSPACE = PSPACE can be reduced to QSAT. □

cont.

Instead we want to use only one copy of ψ_i so instead we have this expression $\psi_{i+1} = \exists z \forall x \forall y [((x = a \wedge y = z) \vee (x = z \wedge y = b)) \rightarrow \psi_i(a, y)]$ where x, y, z are blocks of n^k variables. Now that we have a valid definition of each ϕ_i we now need to get them into a from recognizable by QSAT. First we need to get it into prenex normal form however the quantifiers of ψ_i can simply be migrated to the front behind those of ψ_{i+1}. However the expression still needs to be converted to CNF form, however the space requirements are large so instead a conversion to DNF will be made. This can be done using the DNF of ψ_i followed by $16n^{2k}$ implicants. For integers $1 \leq i, j \leq n^k$ there are 16 implicants of the form $(x_i \wedge \overline{a_i} \wedge x_j \wedge \overline{z_j})$ each corresponding to a way to make $((x = a \wedge y = z) \vee (x = z \wedge y = b))$ false. To obtain ψ_{n^k} we add to ψ_0, n^k sets of implicants and prefix the result with n^k layers of quantifiers. Thus any problem in PSPACE can be reduced to coQSAT and so ant problem in coPSPACE = PSPACE can be reduced to QSAT. □

cont.

Instead we want to use only one copy of ψ_i so instead we have this expression $\psi_{i+1} = \exists z \forall x \forall y[((x = a \wedge y = z) \vee (x = z \wedge y = b)) \rightarrow \psi_i(a, y)]$ where x, y, z are blocks of n^k variables. Now that we have a valid definition of each ϕ_i we now need to get them into a from recognizable by QSAT. First we need to get it into prenex normal form however the quantifiers of ψ_i can simply be migrated to the front behind those of ψ_{i+1}. However the expression still needs to be converted to CNF form, however the space requirements are large so instead a conversion to DNF will be made. This can be done using the DNF of ψ_i followed by $16n^{2k}$ implicants. For integers $1 \leq i, j \leq n^k$ there are 16 implicants of the form $(x_i \wedge \overline{a_i} \wedge x_j \wedge \overline{z_j})$ each corresponding to a way to make $((x = a \wedge y = z) \vee (x = z \wedge y = b))$ false. To obtain ψ_{n^k} we add to ψ_0, n^k sets of implicants and prefix the result with n^k layers of quantifiers. Thus any problem in PSPACE can be reduced to coQSAT and so ant problem in coPSPACE = PSPACE can be reduced to QSAT. □

Outline

Theorem

QSAT is AP-complete

Proof.

It is clear that QSAT is in AP.
To show that it is AP-complete a variation of Cook's theorem is used to capture the computation of a machine which accepts $L \in AP$. The only difference is that the nondeterministic state is universal if the current state is in K_AND and existential otherwise. The alternating Turing Machine can be standardized so that successors of states in K_AND are in K_OR and vice versa. By the addition of padding variables to ensure strict quantifier alternation the resultant expression is a QSAT expression satisfied iff the corresponding input is accepted by M. □

Theorem

AP = PSPACE

Proof.

As both AP and PSPACE are closed under reductions and as they share a complete problem they are equivalent. □

Theorem

QSAT is AP-complete

Proof.

It is clear that QSAT is in AP.
To show that it is AP-complete a variation of Cook's theorem is used to capture the computation of a machine which accepts $L \in AP$. The only difference is that the nondeterministic state is universal if the current state is in K_AND and existential otherwise. The alternating Turing Machine can be standardized so that successors of states in K_AND are in K_OR and vice versa. By the addition of padding variables to ensure strict quantifier alternation the resultant expression is a QSAT expression satisfied iff the corresponding input is accepted by M. □

Theorem

$AP = PSPACE$

Proof.

As both AP and PSPACE are closed under reductions and as they share a complete problem they are equivalent. □

Theorem

QSAT is AP-complete

Proof.

It is clear that QSAT is in AP.
To show that it is AP-complete a variation of Cook's theorem is used to capture the computation of a machine which accepts $L \in AP$. The only difference is that the nondeterministic state is universal if the current state is in K_AND and existential otherwise. The alternating Turing Machine can be standardized so that successors of states in K_AND are in K_OR and vice versa. By the addition of padding variables to ensure strict quantifier alternation the resultant expression is a QSAT expression satisfied iff the corresponding input is accepted by M. □

Theorem

AP = PSPACE

Proof.

As both AP and PSPACE are closed under reductions and as they share a complete problem they are equivalent. □

Theorem

QSAT is AP-complete

Proof.

It is clear that QSAT is in AP.
To show that it is AP-complete a variation of Cook's theorem is used to capture the computation of a machine which accepts $L \in AP$. The only difference is that the nondeterministic state is universal if the current state is in K_AND and existential otherwise. The alternating Turing Machine can be standardized so that successors of states in K_AND are in K_OR and vice versa. By the addition of padding variables to ensure strict quantifier alternation the resultant expression is a QSAT expression satisfied iff the corresponding input is accepted by M. □

Theorem

AP = PSPACE

Proof.

As both AP and PSPACE are closed under reductions and as they share a complete problem they are equivalent. □

Theorem

QSAT is AP-complete

Proof.

It is clear that QSAT is in AP.
To show that it is AP-complete a variation of Cook's theorem is used to capture the computation of a machine which accepts $L \in AP$. The only difference is that the nondeterministic state is universal if the current state is in $K_A ND$ and existential otherwise. The alternating Turing Machine can be standardized so that successors of states in $K_A ND$ are in $K_O R$ and vice versa. By the addition of padding variables to ensure strict quantifier alternation the resultant expression is a QSAT expression satisfied iff the corresponding input is accepted by M. □

Theorem

AP = PSPACE

Proof.

As both AP and PSPACE are closed under reductions and as they share a complete problem they are equivalent. □

Theorem

QSAT is AP-complete

Proof.

It is clear that QSAT is in AP.
To show that it is AP-complete a variation of Cook's theorem is used to capture the computation of a machine which accepts $L \in AP$. The only difference is that the nondeterministic state is universal if the current state is in K_AND and existential otherwise. The alternating Turing Machine can be standardized so that successors of states in K_AND are in K_OR and vice versa. By the addition of padding variables to ensure strict quantifier alternation the resultant expression is a QSAT expression satisfied iff the corresponding input is accepted by M. □

Theorem

AP = PSPACE

Proof.

As both AP and PSPACE are closed under reductions and as they share a complete problem they are equivalent. □

Theorem

QSAT is AP-complete

Proof.

It is clear that QSAT is in AP.
To show that it is AP-complete a variation of Cook's theorem is used to capture the computation of a machine which accepts $L \in AP$. The only difference is that the nondeterministic state is universal if the current state is in K_AND and existential otherwise. The alternating Turing Machine can be standardized so that successors of states in K_AND are in K_OR and vice versa. By the addition of padding variables to ensure strict quantifier alternation the resultant expression is a QSAT expression satisfied iff the corresponding input is accepted by M. □

Theorem

AP = PSPACE

Proof.

As both AP and PSPACE are closed under reductions and as they share a complete problem they are equivalent. □

Outline

1. The complexity class DP
   - Definition of DP
   - Problems in DP
2. The classes P^{NP} and FP^{NP}
   - The definition of P^{NP} and FP^{NP}
3. The polynomial Hierarchy
   - The definition of the Polynomial Hierarchy
   - Examining the Polynomial Hierarchy
   - Diagram of the complexity classes
4. A look a PSPACE
   - QSAT is PSPACE complete
   - PSPACE=AP
   - Geography is PSPACE-complete

What is Geography

Geography is a 2-player game in which players take turns naming cities, with a pre-specified starting city. Each city named has to start with the last letter in the name of the previous cities, and cities cannot be named twice. Any player who is unable to name a valid city looses.

Example

A valid chain of named cities is as follows.
Athens, Syracuse, El Paso, ...

Geography as a decision problem

For a given set C of cities does player 1 have a winning strategy. IE, is there a city player 1 can pick such that no matter what city player 2 picks, there is a city player 1 can pick such that ... player 1 wins.

Geography as a graph

For each city $c \in C$ there is a node v_c in the graph G. Given nodes v_{c_1} and v_{c_2} in G, there is an edge from v_{c_1} to v_{c_2} if city c_2 begins with the last letter of c_2.

Generalization

Thus a generalized version of the problem can be considered as follows, given a directed graph $G(V, E)$ and a starting node v_0 if players take turns selecting edges to form a simple path can player 1 force player 2 to select an edge that forms a cycle.

Theorem

Geography is PSPACE complete.

Proof.

Part 1: Geography$\in$PSPACE. Construct from an instance of Geography a "game tree" where each node in the tree represents a possible state of the game and two nodes are connected if there is a move which gets you from one state to the other. Each leaf node in the tree is then given a value of 1 or 0 depending on whether player 1 wins or loses. And each remaining node is treated as an and gate if it's player 2's move or an or gate if it's player 1's move. As this tree has depth $|V|$ it can be evaluated in polynomial space one branch at a time. □

Proof.

Part 2: Geography if PSPACE complete. We will show this by reducing QSAT to Geography. A QSAT formula $\psi = \exists x_1 \forall x_2 \ldots Q x_n \phi(x_1, x_2, \ldots, x_n)$ is converted to a graph G as follows. Each variable x_i is converted to a choice widget, these ar then chained such that player 1 makes a choice for x_1, player 2 makes a choice for x_2, and so on. The last widget is then connected to a set of nodes, one for each clause, and each of these is connected to some of the other widgets such that if that clause is not satisfied by the choices for $x_1, x_2, \ldots x_n$ then any path from that node leads to an already chosen node. Thus if $\phi \in QSAT$ then there exists a choice for x_1 such that for all choices of x_2 such that ... for all clauses l in ϕ, l is satisfied. Meaning that, by construction, $G \in Geography$. Similarly if $\psi \notin QSAT$, $G \notin Geography$. □