Computational Complexity

Lecture 3

in which we come across

Diagonalization and Time-hierarchies

(But first some more of NP-completeness)

NP-Complete Languages

NP-Complete Languages

- A language L_1 is NP-complete if L_1 is in NP and any NP language L can be reduced to L_1 (Karp reduction: polynomial time many-one reduction)

NP-Complete Languages

- A language L_1 is NP-complete if L_1 is in NP and any NP language L can be reduced to L_1 (Karp reduction: polynomial time many-one reduction)
  - Trivial example: L_1 = TMSAT

NP-Complete Languages

- A language L_1 is NP-complete if L_1 is in NP and any NP language L can be reduced to L_1 (Karp reduction: polynomial time many-one reduction)
  - Trivial example: L_1 = TMSAT
  - First interesting example: L_1 = CKT-SAT

NP-Complete Languages

- A language L_1 is NP-complete if L_1 is in NP and any NP language L can be reduced to L_1 (Karp reduction: polynomial time many-one reduction)
  - Trivial example: L_1 = TMSAT
  - First interesting example: L_1 = CKT-SAT
    - Convert x into a circuit C s.t. C is satisfiable iff x in L

NP-Complete Languages

- A language L_1 is NP-complete if L_1 is in NP and any NP language L can be reduced to L_1 (Karp reduction: polynomial time many-one reduction)
  - Trivial example: L_1 = TMSAT
  - First interesting example: L_1 = CKT-SAT
    - Convert x into a circuit C s.t. C is satisfiable iff x in L
  - More examples, bootstrapping from CKT-SAT

NP-Complete Languages

- A language L_1 is NP-complete if L_1 is in NP and any NP language L can be reduced to L_1 (Karp reduction: polynomial time many-one reduction)
  - Trivial example: L_1 = TMSAT
  - First interesting example: L_1 = CKT-SAT
    - Convert x into a circuit C s.t. C is satisfiable iff x in L
  - More examples, bootstrapping from CKT-SAT
    - If $L \leq_p L_1$ and $L_1 \leq_p L_2$, then $L \leq_p L_2$

CKT-SAT $\leq_p$ SAT

CKT-SAT $\leq_p$ SAT

- SAT: Are all given "clauses" simultaneously satisfiable? (Conjunctive Normal Form)

CKT-SAT $\leq_p$ SAT

- SAT: Are all given "clauses" simultaneously satisfiable? (Conjunctive Normal Form)
- Converting a circuit to a collection of clauses:

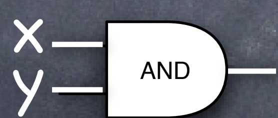
CKT-SAT $\leq_p$ SAT

- SAT: Are all given "clauses" simultaneously satisfiable? (Conjunctive Normal Form)
- Converting a circuit to a collection of clauses:
    - For each wire (connected component), add a variable

CKT-SAT $\leq_p$ SAT

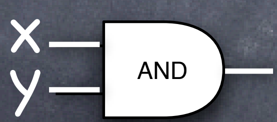
- SAT: Are all given "clauses" simultaneously satisfiable? (Conjunctive Normal Form)
- Converting a circuit to a collection of clauses:
  - For each wire (connected component), add a variable
  - Add output variable as a clause. And for each gate, add a clause involving variables for wires connected to the gate:

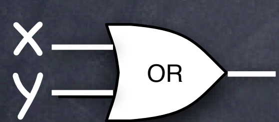
CKT-SAT $\leq_p$ SAT

- SAT: Are all given "clauses" simultaneously satisfiable? (Conjunctive Normal Form)
- Converting a circuit to a collection of clauses:
  - For each wire (connected component), add a variable
  - Add output variable as a clause. And for each gate, add a clause involving variables for wires connected to the gate:
    - e.g. 

z: $(z \Rightarrow x)$, $(z \Rightarrow y)$, $(\neg z \Rightarrow \neg x \vee \neg y)$.
i.e., $(\neg z \vee x)$, $(\neg z \vee y)$, $(z \vee \neg x \vee \neg y)$.

CKT-SAT $\leq_p$ SAT

- SAT: Are all given "clauses" simultaneously satisfiable? (Conjunctive Normal Form)
- Converting a circuit to a collection of clauses:
  - For each wire (connected component), add a variable
  - Add output variable as a clause. And for each gate, add a clause involving variables for wires connected to the gate:
    - e.g. 

z: $(z \Rightarrow x)$, $(z \Rightarrow y)$, $(\neg z \Rightarrow \neg x \vee \neg y)$.
i.e., $(\neg z \vee x)$, $(\neg z \vee y)$, $(z \vee \neg x \vee \neg y)$.
    - and 

z: $(z \Rightarrow x \vee y)$, $(\neg z \Rightarrow \neg x)$, $(\neg z \Rightarrow \neg y)$.

SAT $\leq_p$ 3SAT

SAT $\leq_p$ 3SAT

- Previous reduction was to 3SAT, so 3SAT is NP-complete. And SAT is in NP. So SAT $\leq_p$ 3SAT.

SAT $\leq_p$ 3SAT

- Previous reduction was to 3SAT, so 3SAT is NP-complete. And SAT is in NP. So SAT $\leq_p$ 3SAT.

- More directly:

SAT $\leq_p$ 3SAT

- Previous reduction was to 3SAT, so 3SAT is NP-complete. And SAT is in NP. So SAT $\leq_p$ 3SAT.
- More directly:
  - $(a \vee b \vee c \vee d \vee e) \rightarrow (a \vee b \vee x), (\neg x \vee c \vee d \vee e)$

    $\rightarrow (a \vee b \vee x), (\neg x \vee c \vee y), (\neg y \vee d \vee e)$

SAT $\leq_p$ 3SAT

- Previous reduction was to 3SAT, so 3SAT is NP-complete. And SAT is in NP. So SAT $\leq_p$ 3SAT.
- More directly:
  - $(a \lor b \lor c \lor d \lor e) \rightarrow (a \lor b \lor x), (\neg x \lor c \lor d \lor e)$

    $\rightarrow (a \lor b \lor x), (\neg x \lor c \lor y), (\neg y \lor d \lor e)$
- Reduction needs 3SAT

SAT $\leq_p$ 3SAT

- Previous reduction was to 3SAT, so 3SAT is NP-complete. And SAT is in NP. So SAT $\leq_p$ 3SAT.
- More directly:
  - $(a \vee b \vee c \vee d \vee e) \rightarrow (a \vee b \vee x), (\neg x \vee c \vee d \vee e)$

    $\rightarrow (a \vee b \vee x), (\neg x \vee c \vee y), (\neg y \vee d \vee e)$
- Reduction needs 3SAT
  - 2SAT is in fact in P! [Exercise]

SAT $\leq_p$ 3SAT

- Previous reduction was to 3SAT, so 3SAT is NP-complete. And SAT is in NP. So SAT $\leq_p$ 3SAT.
- More directly:
  - $(a \vee b \vee c \vee d \vee e) \rightarrow (a \vee b \vee x), (\neg x \vee c \vee d \vee e)$
    $\rightarrow (a \vee b \vee x), (\neg x \vee c \vee y), (\neg y \vee d \vee e)$
- Reduction needs 3SAT
  - 2SAT is in fact in P! [Exercise]
- Reduction not parsimonious (can you make it? [Exercise])

3SAT $\leq_p$ CLIQUE

3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?

$3SAT \leq_p CLIQUE$

- CLIQUE: Does graph G have a clique of size m?

$(x \lor \neg y \lor \neg z)$

$(w \lor y)$

$(w \lor x \lor \neg z)$

3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph

$(x \vee \neg y \vee \neg z)$

$(w \vee y)$

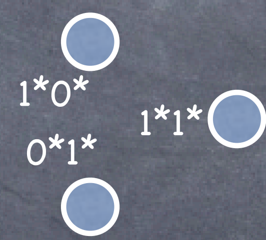
$(w \vee x \vee \neg z)$

3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph

$(x \vee \neg y \vee \neg z)$

$(w \vee y)$



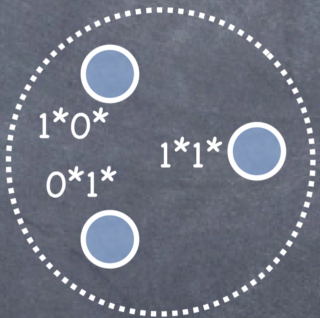
$(w \vee x \vee \neg z)$

3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph

$(x \vee \neg y \vee \neg z)$

$(w \vee y)$

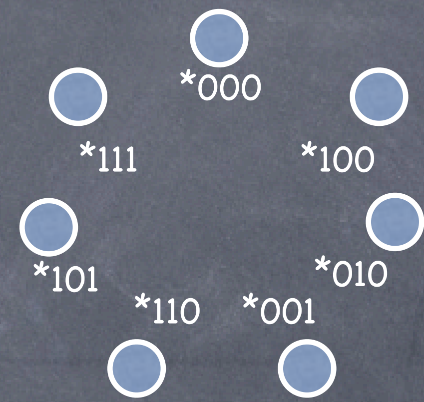


$(w \vee x \vee \neg z)$

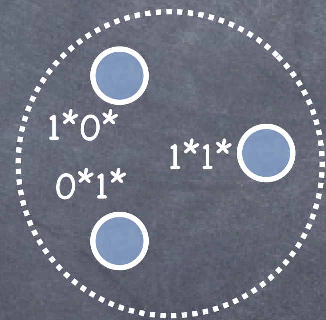
3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph

$(x \vee \neg y \vee \neg z)$



$(w \vee y)$

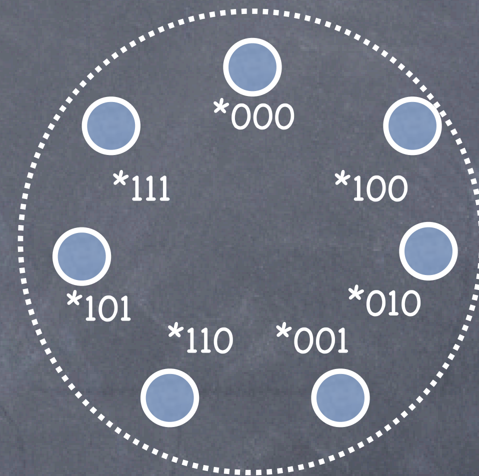


$(w \vee x \vee \neg z)$

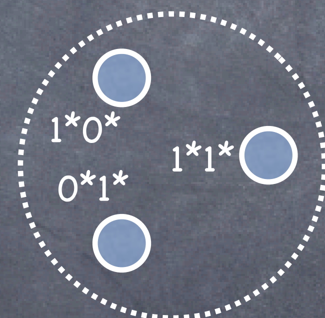
3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph

$(x \vee \neg y \vee \neg z)$



$(w \vee y)$

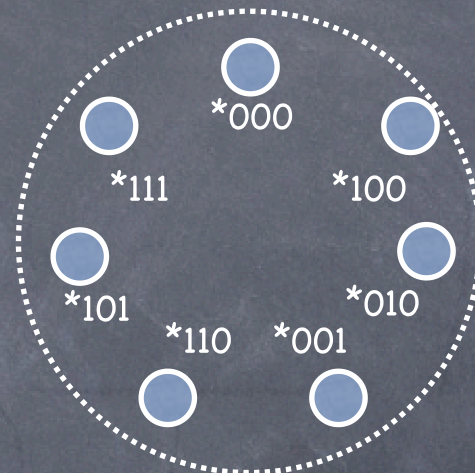


$(w \vee x \vee \neg z)$

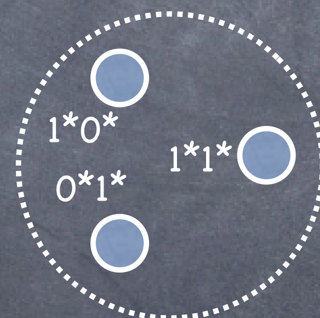
3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph

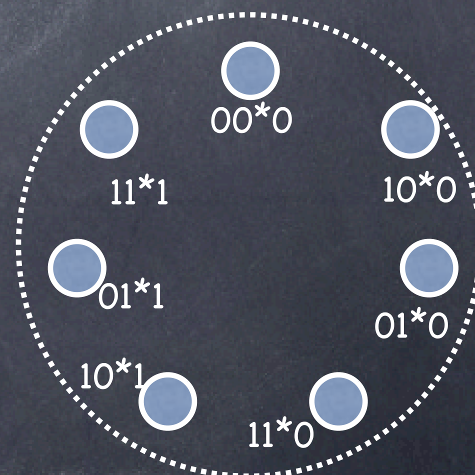
$(x \lor \neg y \lor \neg z)$



$(w \lor y)$



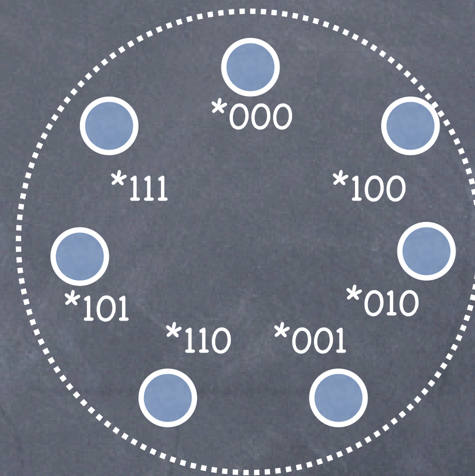
$(w \lor x \lor \neg z)$



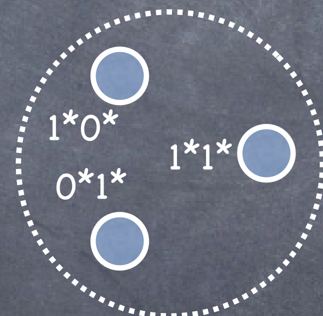
3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)

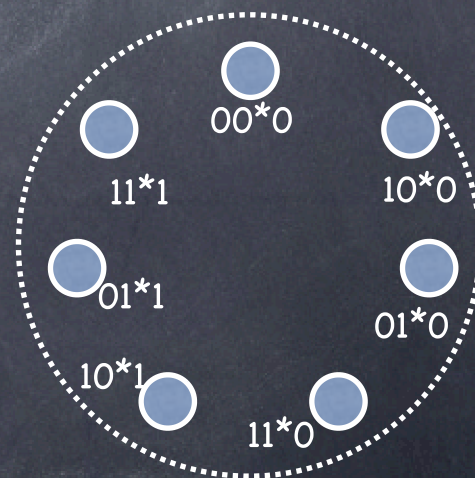
$(x \vee \neg y \vee \neg z)$



$(w \vee y)$

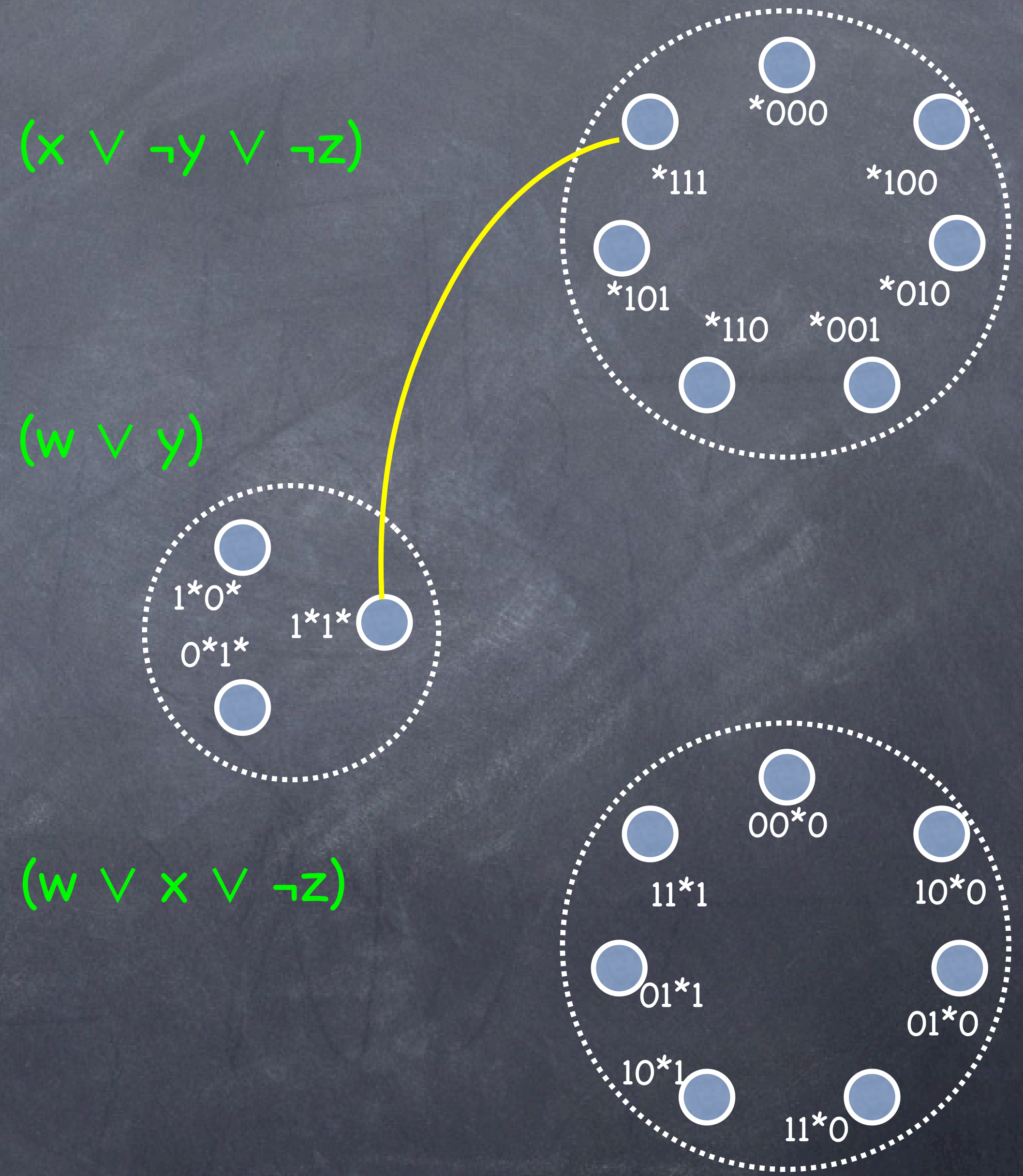


$(w \vee x \vee \neg z)$



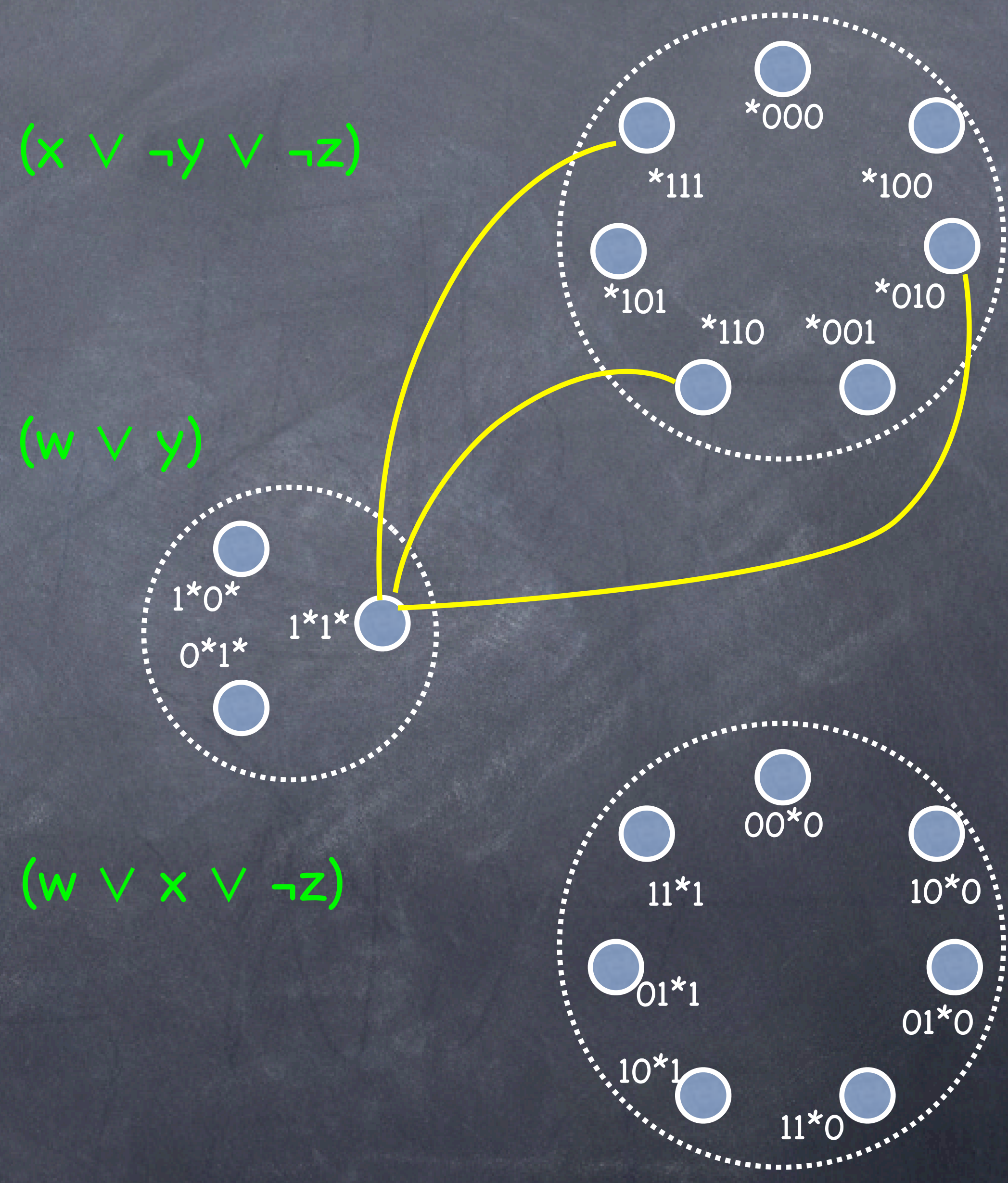
3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)



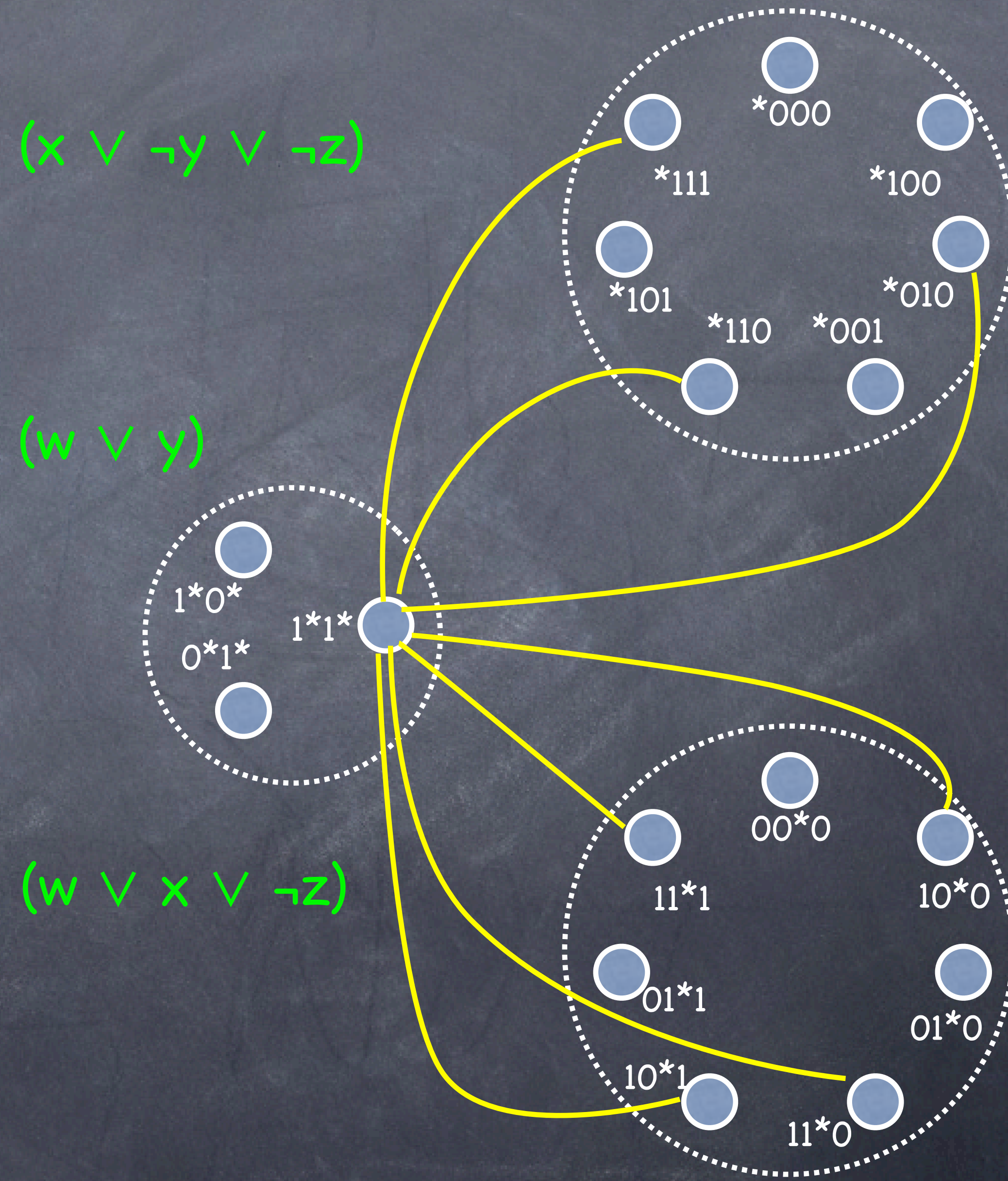
3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)



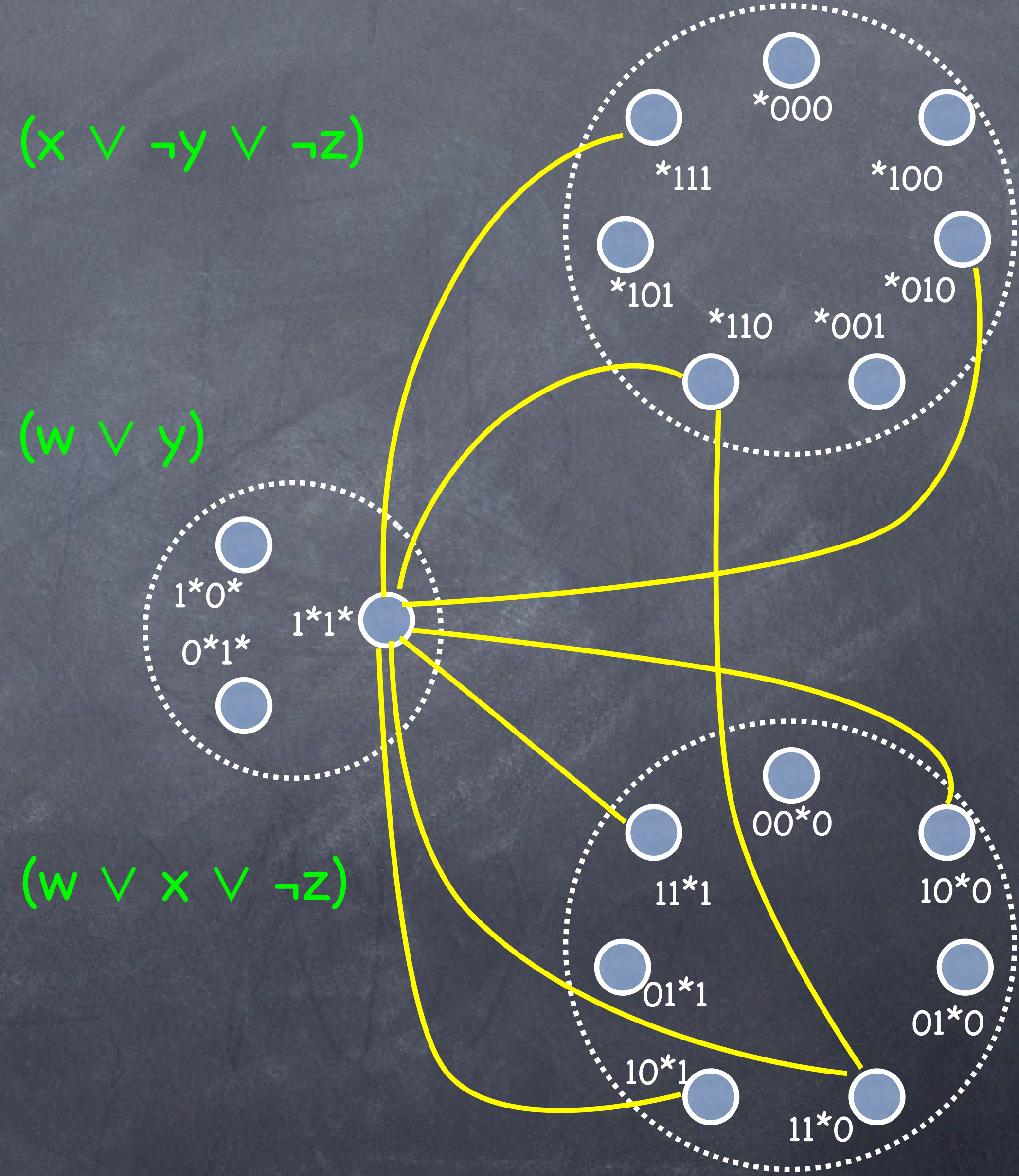
3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)



3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)



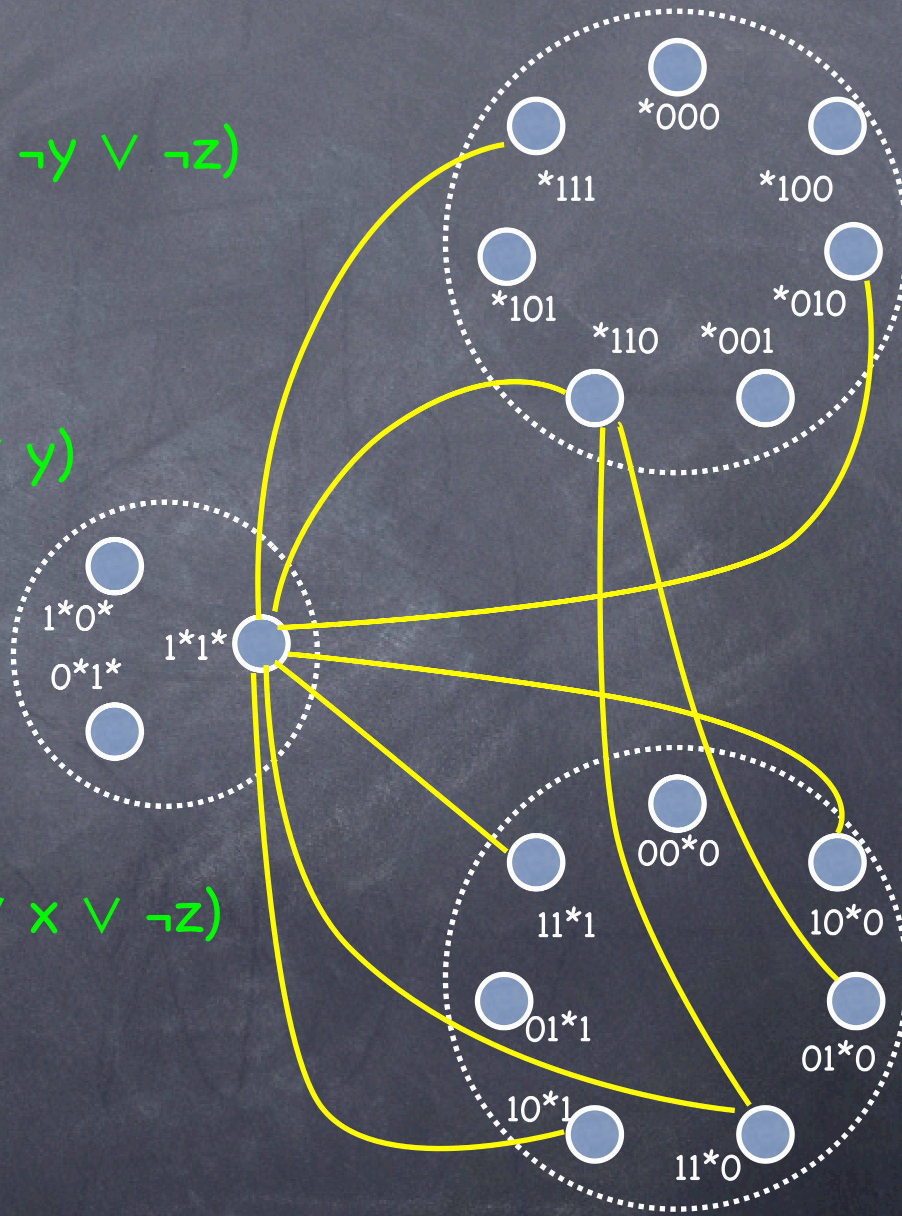
3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)

$(x \vee \neg y \vee \neg z)$

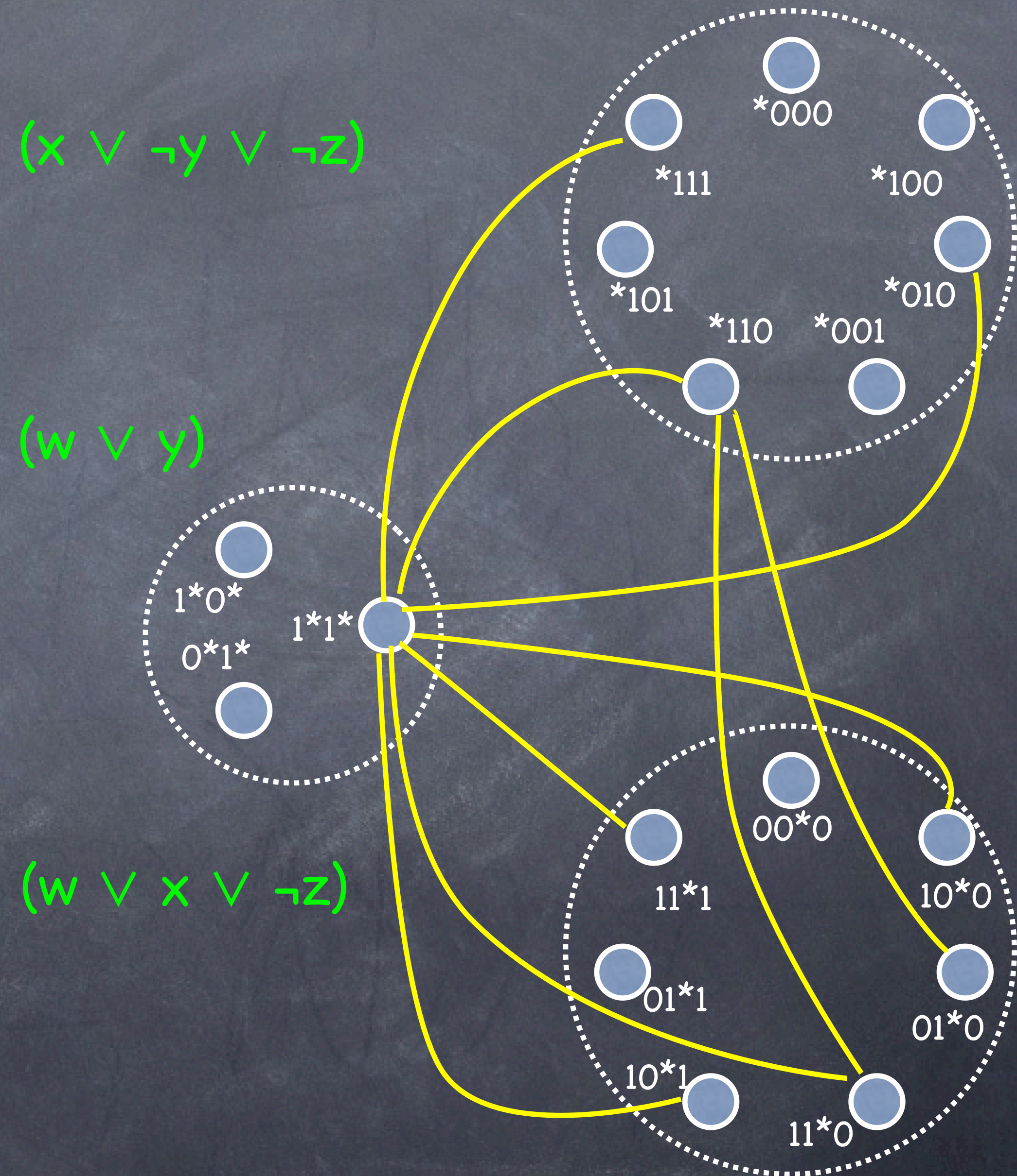
$(w \vee y)$

$(w \vee x \vee \neg z)$



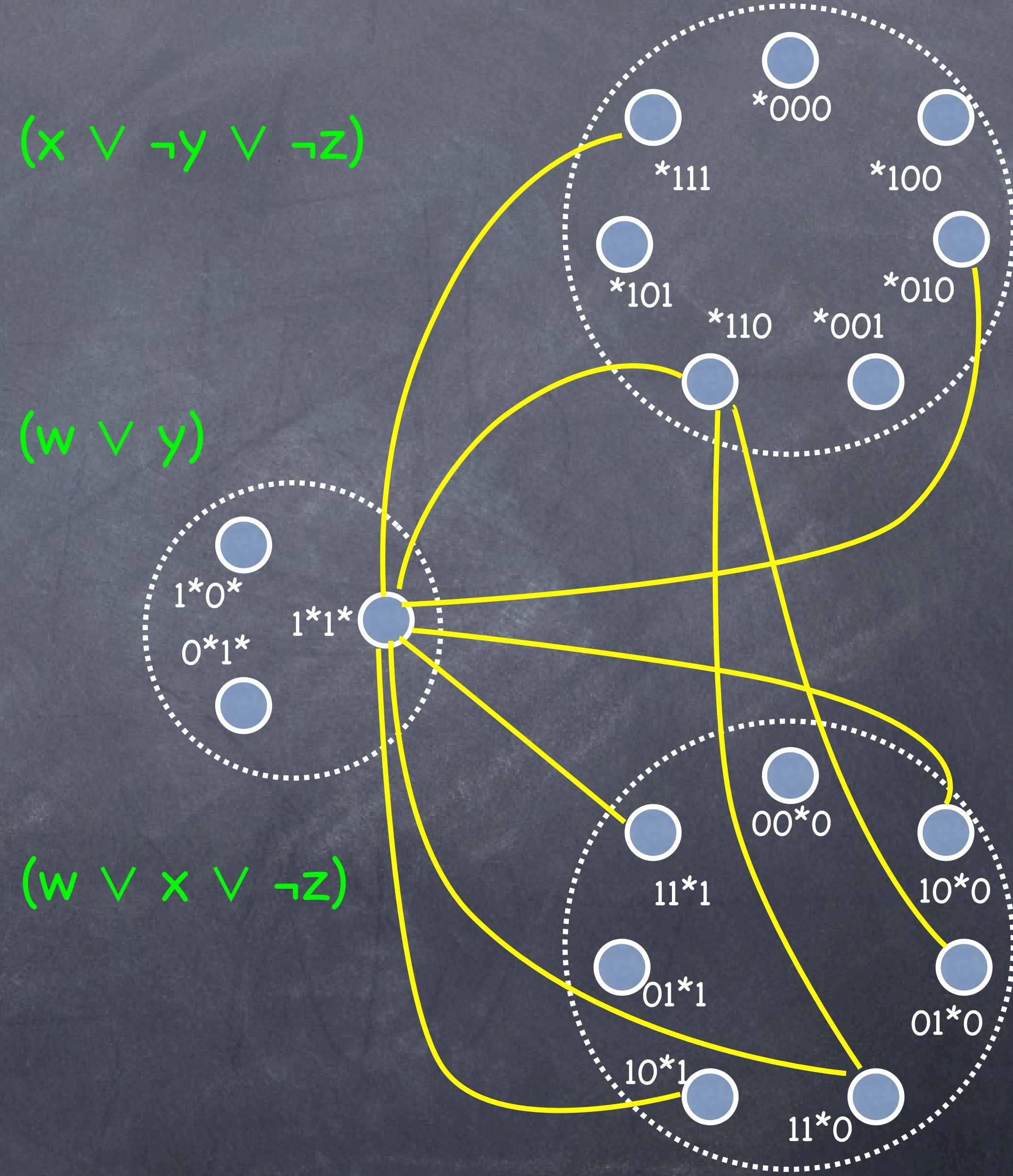
3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments



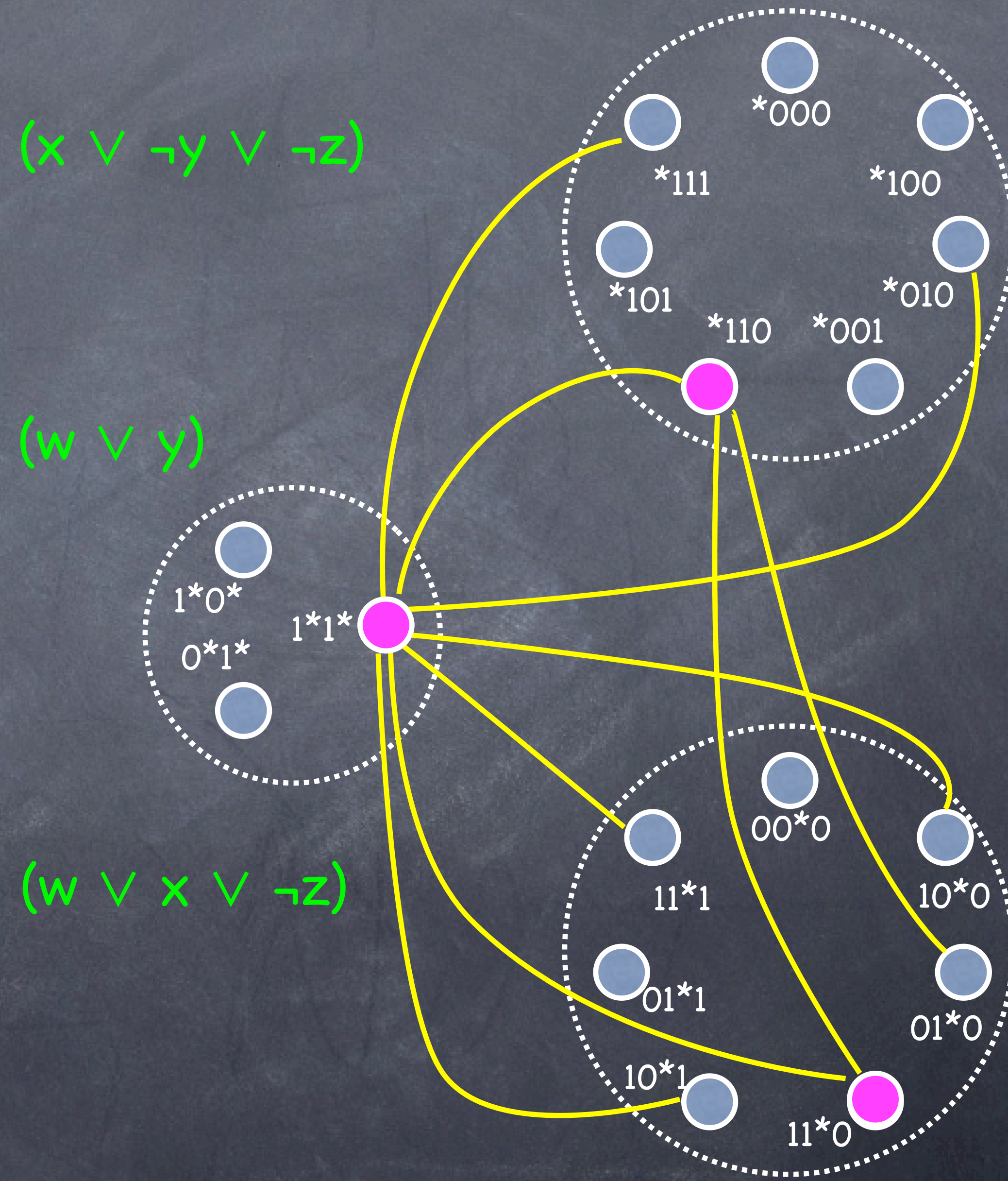
3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments
  - m-clique iff all m clauses satisfiable



3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments
  - m-clique iff all m clauses satisfiable



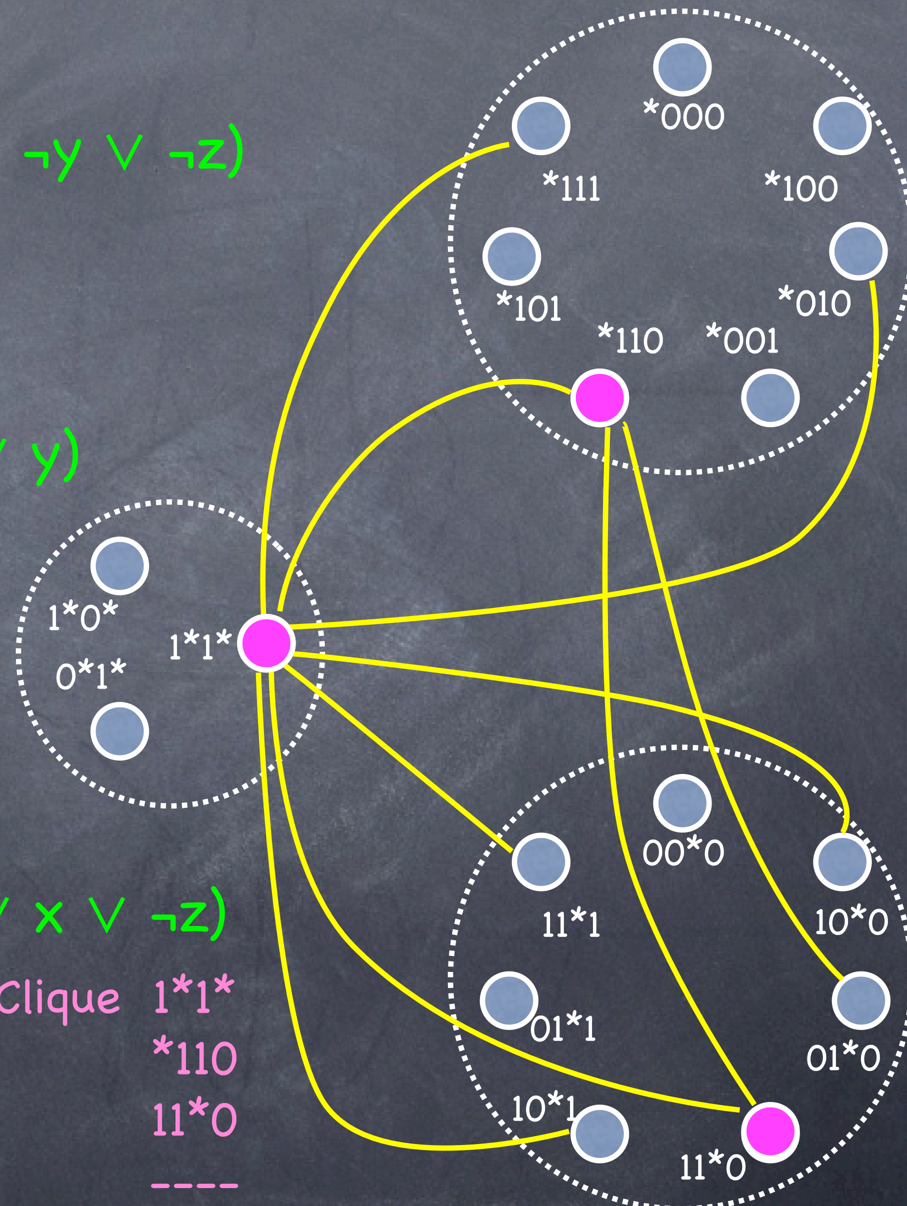
3SAT $\leq_p$ CLIQUE

- CLIQUE: Does graph G have a clique of size m?
- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments
  - m-clique iff all m clauses satisfiable

$(x \vee \neg y \vee \neg z)$

$(w \vee y)$

$(w \vee x \vee \neg z)$



3-Clique 1*1*
*110
11*0
----
sat assignment 1110

INDEP-SET and VERTEX-COVER

INDEP-SET and VERTEX-COVER

- CLIQUE $\leq_p$ INDEP-SET

INDEP-SET and VERTEX-COVER

- CLIQUE $\leq_p$ INDEP-SET
  - G has an m-clique iff G^c has an m-independent-set

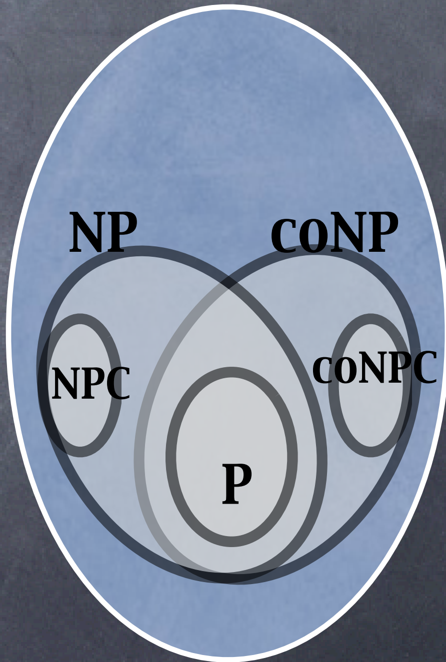
INDEP-SET and VERTEX-COVER

- CLIQUE $\leq_p$ INDEP-SET
  - G has an m-clique iff G^c has an m-independent-set
- INDEP-SET $\leq_p$ VERTEX-COVER

INDEP-SET and VERTEX-COVER

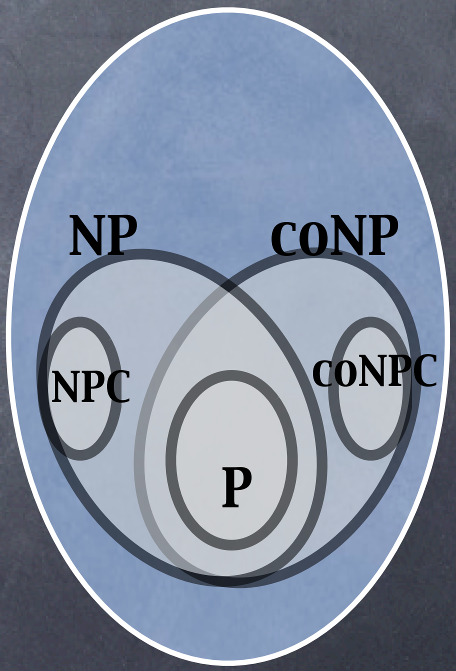
- CLIQUE $\leq_p$ INDEP-SET
  - G has an m-clique iff G^c has an m-independent-set
- INDEP-SET $\leq_p$ VERTEX-COVER
  - G has an m-indep-set iff G has an (n-m)-vertex-cover

NP, P, co-NP and NPC



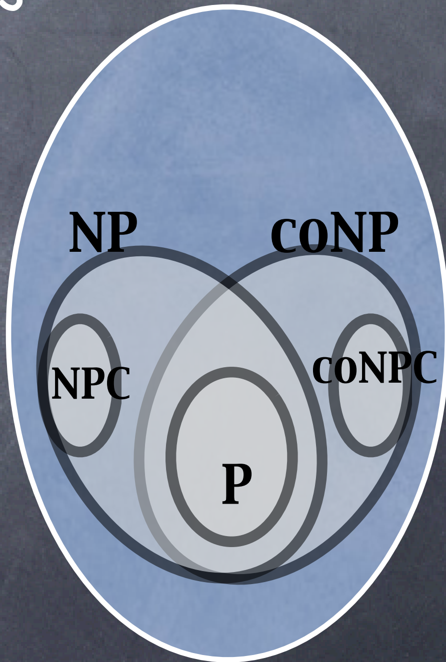
NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X



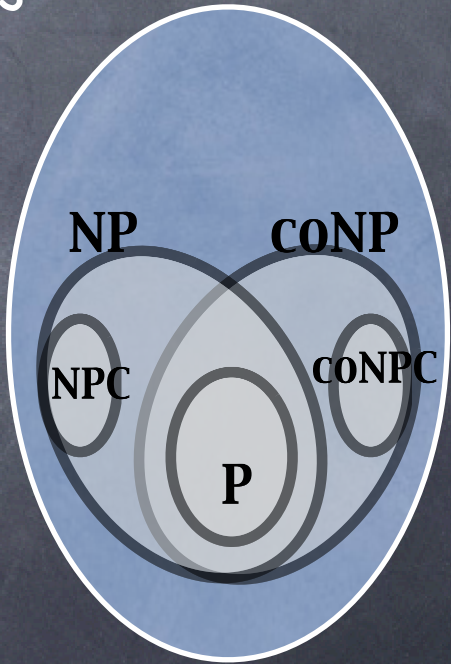
NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X
    - e.g. P, NP are closed under polynomial reductions



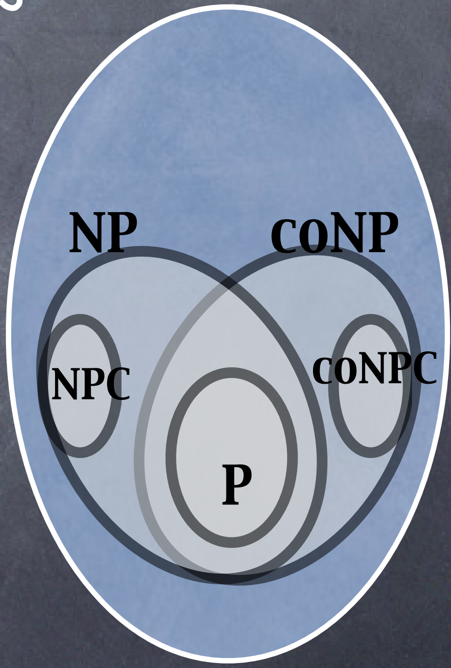
NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X
  - e.g. P, NP are closed under polynomial reductions
    - So is co-NP (If X is closed, so is co-X. Why?)



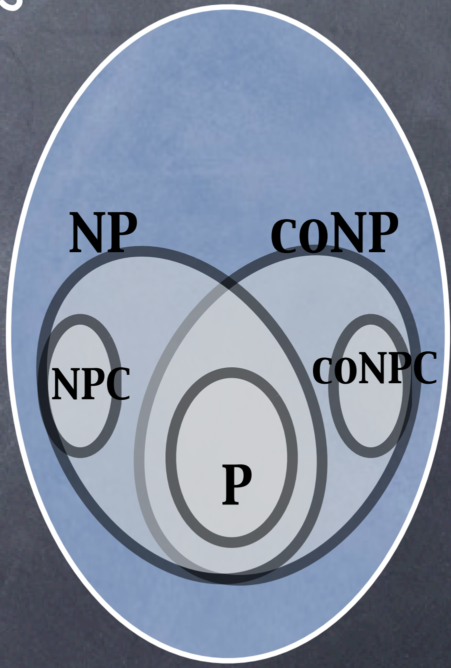
NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X
  - e.g. P, NP are closed under polynomial reductions
    - So is co-NP (If X is closed, so is co-X. Why?)
- If any NPC language is in P, then NP = P



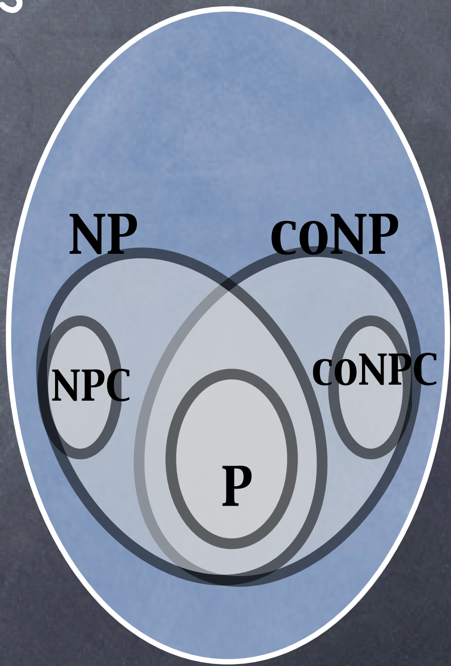
NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X
  - e.g. P, NP are closed under polynomial reductions
    - So is co-NP (If X is closed, so is co-X. Why?)
- If any NPC language is in P, then NP = P
- If any NPC language is in co-NP, then NP = co-NP



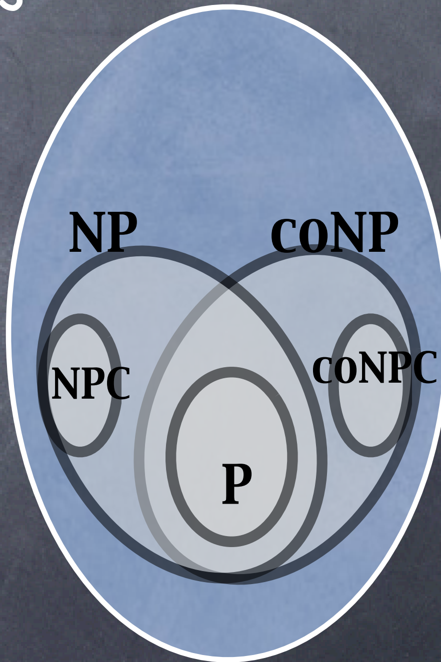
NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X
  - e.g. P, NP are closed under polynomial reductions
    - So is co-NP (If X is closed, so is co-X. Why?)
- If any NPC language is in P, then NP = P
- If any NPC language is in co-NP, then NP = co-NP
  - Note: $X \subseteq$ co-X $\Rightarrow$ X = co-X (Why?)



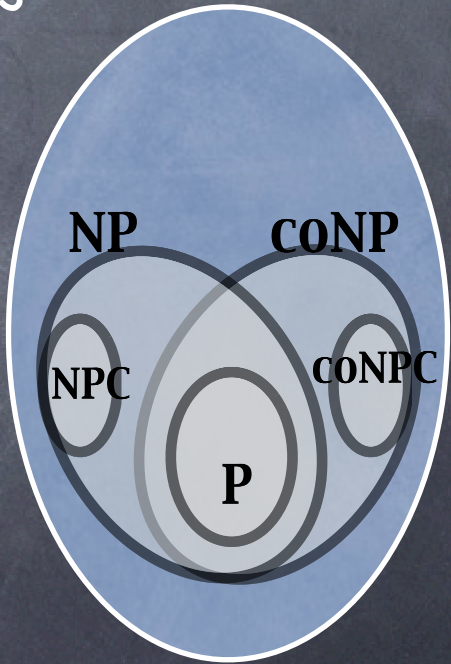
NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X
  - e.g. P, NP are closed under polynomial reductions
    - So is co-NP (If X is closed, so is co-X. Why?)
- If any NPC language is in P, then NP = P
- If any NPC language is in co-NP, then NP = co-NP
  - Note: $X \subseteq \text{co-}X \Rightarrow X = \text{co-}X$ (Why?)
- L is NP-complete iff L^c is co-NP-complete (Why?)



NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X
  - e.g. P, NP are closed under polynomial reductions
    - So is co-NP (If X is closed, so is co-X. Why?)
- If any NPC language is in P, then NP = P
- If any NPC language is in co-NP, then NP = co-NP
  - Note: $X \subseteq \text{co-}X \Rightarrow X = \text{co-}X$ (Why?)
- L is NP-complete iff L^c is co-NP-complete (Why?)
  - co-NP complete = co-(NP-complete)



Separating Classes

Separating Classes

- How to prove a set X strictly bigger than Y

Separating Classes

- How to prove a set X strictly bigger than Y
  - Show an element not in Y, but in X? For us, not in Y may often be difficult to prove for (familiar) elements

Separating Classes

- How to prove a set X strictly bigger than Y
  - Show an element not in Y, but in X? For us, not in Y may often be difficult to prove for (familiar) elements
  - Count? What if both infinite?!

Separating Classes

- How to prove a set X strictly bigger than Y
  - Show an element not in Y, but in X? For us, not in Y may often be difficult to prove for (familiar) elements
  - Count? What if both infinite?!
  - Comparing infinite sets: diagonalization!

Cantor’s Diagonal Slash

Cantor's Diagonal Slash

- Are real numbers (say in the range [0,1)) countable?

Cantor's Diagonal Slash

- Are real numbers (say in the range [0,1)) countable?
  - Suppose they were: consider enumerating them along with their binary representations in a table

Cantor's Diagonal Slash

- Are real numbers (say in the range [0,1)) countable?
  - Suppose they were: consider enumerating them along with their binary representations in a table

R_i									
R_1 =	1	0	0	1	0	0	0	0	1
R_2 =	0	0	1	0	1	0	0	1	1
R_3 =	1	1	1	1	1	1	1	0	0
R_4 =	1	1	0	1	0	1	0	1	1
R_5 =	1	1	0	0	0	0	1	0	0
R_6 =	0	0	0	0	0	0	1	1	0
R_7 =	0	1	0	1	0	1	0	1	1

Cantor's Diagonal Slash

- Are real numbers (say in the range [0,1)) countable?
  - Suppose they were: consider enumerating them along with their binary representations in a table
  - Consider the real number corresponding to the "flipped diagonal"

R_i									
R_1 =	1	0	0	1	0	0	0	0	1
R_2 =	0	0	1	0	1	0	0	1	1
R_3 =	1	1	1	1	1	1	1	0	0
R_4 =	1	1	0	1	0	1	0	1	1
R_5 =	1	1	0	0	0	0	1	0	0
R_6 =	0	0	0	0	0	0	1	1	0
R_7 =	0	1	0	1	0	1	0	1	1

Cantor's Diagonal Slash

- Are real numbers (say in the range [0,1)) countable?
  - Suppose they were: consider enumerating them along with their binary representations in a table
  - Consider the real number corresponding to the "flipped diagonal"

R_i									
R_1 =	1	0	0	1	0	0	0	0	1
R_2 =	0	0	1	0	1	0	0	1	1
R_3 =	1	1	1	1	1	1	1	0	0
R_4 =	1	1	0	1	0	1	0	1	1
R_5 =	1	1	0	0	0	0	1	0	0
R_6 =	0	0	0	0	0	0	1	1	0
R_7 =	0	1	0	1	0	1	0	1	1

Cantor's Diagonal Slash

- Are real numbers (say in the range [0,1)) countable?
  - Suppose they were: consider enumerating them along with their binary representations in a table
  - Consider the real number corresponding to the "flipped diagonal"
    - Doesn't appear in this table!

R_i									
R_1 =	1	0	0	1	0	0	0	0	1
R_2 =	0	0	1	0	1	0	0	1	1
R_3 =	1	1	1	1	1	1	1	0	0
R_4 =	1	1	0	1	0	1	0	1	1
R_5 =	1	1	0	0	0	0	1	0	0
R_6 =	0	0	0	0	0	0	1	1	0
R_7 =	0	1	0	1	0	1	0	1	1

Cantor's Diagonal Slash

This table can't have all reals

- Are real numbers (say in the range [0,1)) countable?
  - Suppose they were: consider enumerating them along with their binary representations in a table
  - Consider the real number corresponding to the "flipped diagonal"
    - Doesn't appear in this table!

R_i									
R_1 =	1	0	0	1	0	0	0	0	1
R_2 =	0	0	1	0	1	0	0	1	1
R_3 =	1	1	1	1	1	1	1	0	0
R_4 =	1	1	0	1	0	1	0	1	1
R_5 =	1	1	0	0	0	0	1	0	0
R_6 =	0	0	0	0	0	0	1	1	0
R_7 =	0	1	0	1	0	1	0	1	1

Undecidable Languages

L_{M1} =	1	0	0	1	0	0	0	0	1
L_{M2} =	0	0	1	0	1	0	0	1	1
L_{M3} =	1	1	1	1	1	1	1	0	0
L_{M4} =	1	1	0	1	0	1	0	1	1
L_{M5} =	1	1	0	0	0	0	1	0	0
L_{M6} =	0	0	0	0	0	0	1	1	0
L_{M7} =	0	1	0	1	0	1	0	1	1

Undecidable Languages

- Languages, like real numbers, can be represented as infinite bit-vectors

L_{M1} =	1	0	0	1	0	0	0	0	1
L_{M2} =	0	0	1	0	1	0	0	1	1
L_{M3} =	1	1	1	1	1	1	1	0	0
L_{M4} =	1	1	0	1	0	1	0	1	1
L_{M5} =	1	1	0	0	0	0	1	0	0
L_{M6} =	0	0	0	0	0	0	1	1	0
L_{M7} =	0	1	0	1	0	1	0	1	1

Undecidable Languages

- Languages, like real numbers, can be represented as infinite bit-vectors
- TMs can be enumerated!

L_{M1} =	1	0	0	1	0	0	0	0	1
L_{M2} =	0	0	1	0	1	0	0	1	1
L_{M3} =	1	1	1	1	1	1	1	0	0
L_{M4} =	1	1	0	1	0	1	0	1	1
L_{M5} =	1	1	0	0	0	0	1	0	0
L_{M6} =	0	0	0	0	0	0	1	1	0
L_{M7} =	0	1	0	1	0	1	0	1	1

Undecidable Languages

- Languages, like real numbers, can be represented as infinite bit-vectors
- TMs can be enumerated!
- Table of languages recognized by the TMs

L_{M1} =	1	0	0	1	0	0	0	0	1
L_{M2} =	0	0	1	0	1	0	0	1	1
L_{M3} =	1	1	1	1	1	1	1	0	0
L_{M4} =	1	1	0	1	0	1	0	1	1
L_{M5} =	1	1	0	0	0	0	1	0	0
L_{M6} =	0	0	0	0	0	0	1	1	0
L_{M7} =	0	1	0	1	0	1	0	1	1

Undecidable Languages

- Languages, like real numbers, can be represented as infinite bit-vectors
- TMs can be enumerated!
- Table of languages recognized by the TMs
- L = "diagonal language"

L_{M1} =	1	0	0	1	0	0	0	0	1
L_{M2} =	0	0	1	0	1	0	0	1	1
L_{M3} =	1	1	1	1	1	1	1	0	0
L_{M4} =	1	1	0	1	0	1	0	1	1
L_{M5} =	1	1	0	0	0	0	1	0	0
L_{M6} =	0	0	0	0	0	0	1	1	0
L_{M7} =	0	1	0	1	0	1	0	1	1

Undecidable Languages

- Languages, like real numbers, can be represented as infinite bit-vectors
- TMs can be enumerated!
- Table of languages recognized by the TMs
- L = "diagonal language"
  - L^c does not appear as a row in this table. Hence not recognizable!

L_{M1} =	1	0	0	1	0	0	0	0	1
L_{M2} =	0	0	1	0	1	0	0	1	1
L_{M3} =	1	1	1	1	1	1	1	0	0
L_{M4} =	1	1	0	1	0	1	0	1	1
L_{M5} =	1	1	0	0	0	0	1	0	0
L_{M6} =	0	0	0	0	0	0	1	1	0
L_{M7} =	0	1	0	1	0	1	0	1	1

Undecidable Languages

- Languages, like real numbers, can be represented as infinite bit-vectors
- TMs can be enumerated!
- Table of languages recognized by the TMs
- L = "diagonal language"
  - L^c does not appear as a row in this table. Hence not recognizable!

This table can't have all languages

L_{M1} =	1	0	0	1	0	0	0	0	1
L_{M2} =	0	0	1	0	1	0	0	1	1
L_{M3} =	1	1	1	1	1	1	1	0	0
L_{M4} =	1	1	0	1	0	1	0	1	1
L_{M5} =	1	1	0	0	0	0	1	0	0
L_{M6} =	0	0	0	0	0	0	1	1	0
L_{M7} =	0	1	0	1	0	1	0	1	1

Diagonalization to Separate Classes

- Diagonalization can separate the class of decidable languages (from the class of all languages)
  - Plan: Use similar techniques to separate complexity classes

DTIME Hierarchy

DTIME Hierarchy

- Fix a TM model (one-tape, binary alphabet)

DTIME Hierarchy

- Fix a TM model (one-tape, binary alphabet)
  - DTIME(T) = class of languages that can be decided in O(T(n)) time, by such a TM

DTIME Hierarchy

- Fix a TM model (one-tape, binary alphabet)
  - DTIME(T) = class of languages that can be decided in O(T(n)) time, by such a TM
- Theorem: $\mathrm{DTIME}(n^c) \subsetneq \mathrm{DTIME}(n^{c+1})$ for all $c \geq 1$

DTIME Hierarchy

- Fix a TM model (one-tape, binary alphabet)
  - DTIME(T) = class of languages that can be decided in O(T(n)) time, by such a TM
- Theorem: $\mathrm{DTIME}(n^c) \subsetneq \mathrm{DTIME}(n^{c+1})$ for all $c \geq 1$
  - More generally $\mathrm{DTIME}(T) \subsetneq \mathrm{DTIME}(T')$ if T, T' "nice" (and $\geq$ n) and $T(n)\log(T(n)) = o(T'(n))$

DTIME Hierarchy

- Fix a TM model (one-tape, binary alphabet)
  - DTIME(T) = class of languages that can be decided in O(T(n)) time, by such a TM
- Theorem: $\mathrm{DTIME}(n^c) \subsetneq \mathrm{DTIME}(n^{c+1})$ for all $c \geq 1$
  - More generally $\mathrm{DTIME}(T) \subsetneq \mathrm{DTIME}(T')$ if T, T' "nice" (and $\geq$ n) and $T(n)\log(T(n)) = o(T'(n))$
- Consequences, for e.g., $P \subsetneq EXP$

DTIME Hierarchy

- Fix a TM model (one-tape, binary alphabet)
  - DTIME(T) = class of languages that can be decided in O(T(n)) time, by such a TM
- Theorem: $\mathrm{DTIME}(n^c) \subsetneq \mathrm{DTIME}(n^{c+1})$ for all $c \geq 1$
  - More generally $\mathrm{DTIME}(T) \subsetneq \mathrm{DTIME}(T')$ if T, T' "nice" (and $\geq n$) and $T(n)\log(T(n)) = o(T'(n))$
- Consequences, for e.g., $P \subsetneq EXP$
  - $P \subseteq \mathrm{DTIME}(2^n) \subsetneq \mathrm{DTIME}(2^{2n}) \subseteq EXP$

DTIME Hierarchy: Proof

DTIME Hierarchy: Proof

- M_i be an enumeration of TMs, each TM appearing infinitely often

DTIME Hierarchy: Proof

- M_i be an enumeration of TMs, each TM appearing infinitely often
- Consider table(i,j) = UTM$|_{T'}$ (M_i,j), where T log T = o(T′)

M_i \ j						
	1	0	0	1	0	0
	0	0	1	0	1	0
	1	1	1	1	1	1
	1	1	0	1	0	1
	1	1	0	0	0	0

DTIME Hierarchy: Proof

- M_i be an enumeration of TMs, each TM appearing infinitely often
- Consider table(i,j) = $UTM|_{T'}$ (M_i,j), where T log T = o(T′)

T′ large and nice enough to allow simulation

M_i \ j					
	1	0	0	1	0
	0	0	1	0	1
	1	1	1	1	1
	1	1	0	1	0
	1	1	0	0	0

DTIME Hierarchy: Proof

- M_i be an enumeration of TMs, each TM appearing infinitely often
- Consider table(i,j) = $UTM|_{T'}(M_i,j)$, where $T \log T = o(T')$

T′ large and nice enough to allow simulation

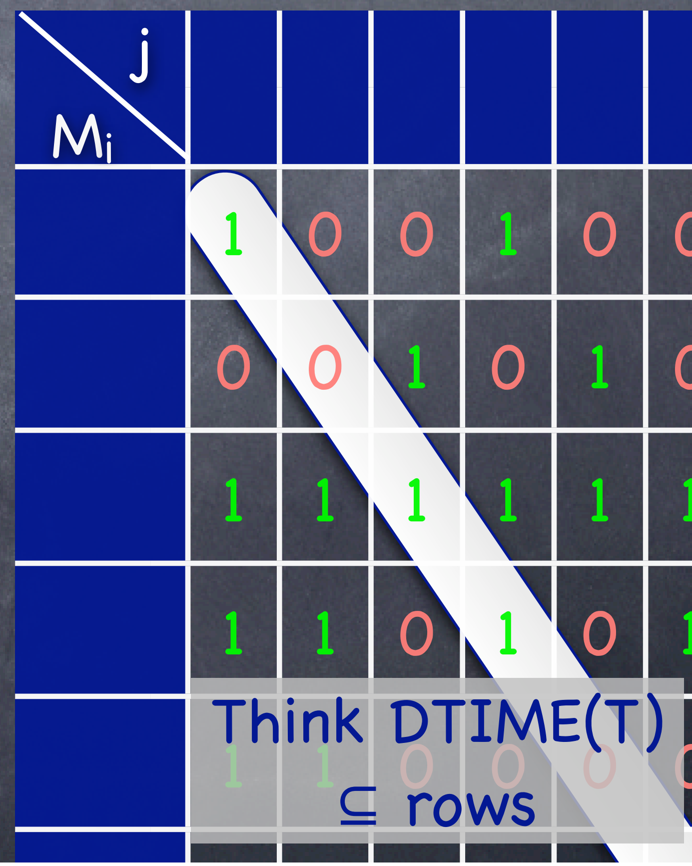
M_i \ j						
	1	0	0	1	0	0
	0	0	1	0	1	0
	1	1	1	1	1	1
	1	1	0	1	0	1
	1	1	0	0	0	0

Think DTIME(T) $\subseteq$ rows

DTIME Hierarchy: Proof

- M_i be an enumeration of TMs, each TM appearing infinitely often
- Consider table(i,j) = $UTM|_{T'}(M_i,j)$, where $T \log T = o(T')$

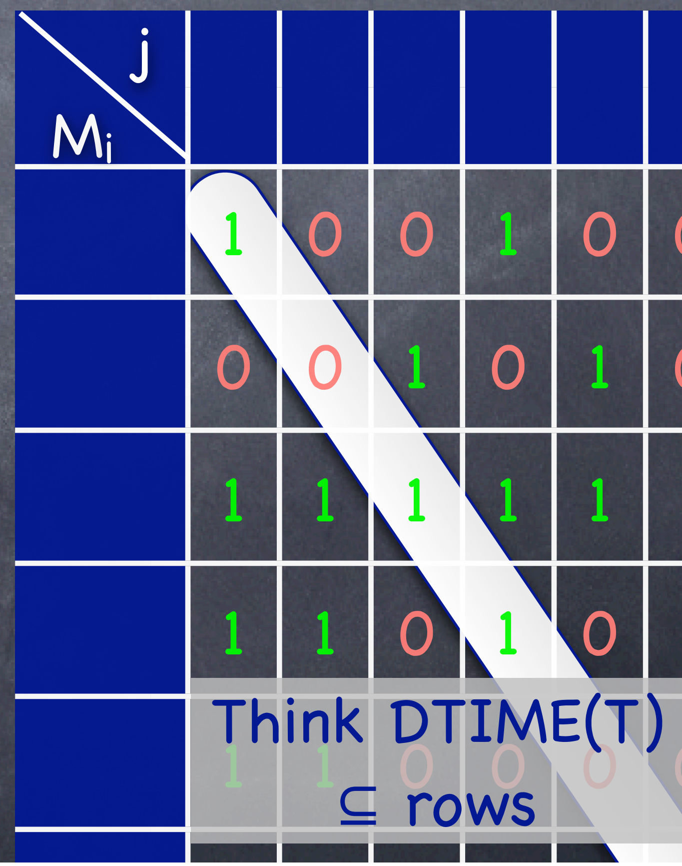
T′ large and nice enough to allow simulation



DTIME Hierarchy: Proof

- M_i be an enumeration of TMs, each TM appearing infinitely often
- Consider table(i,j) = $UTM|_{T'}(M_i,j)$, where $T \log T = o(T')$
- Let L′ = inverted diagonal.

T′ large and nice enough to allow simulation

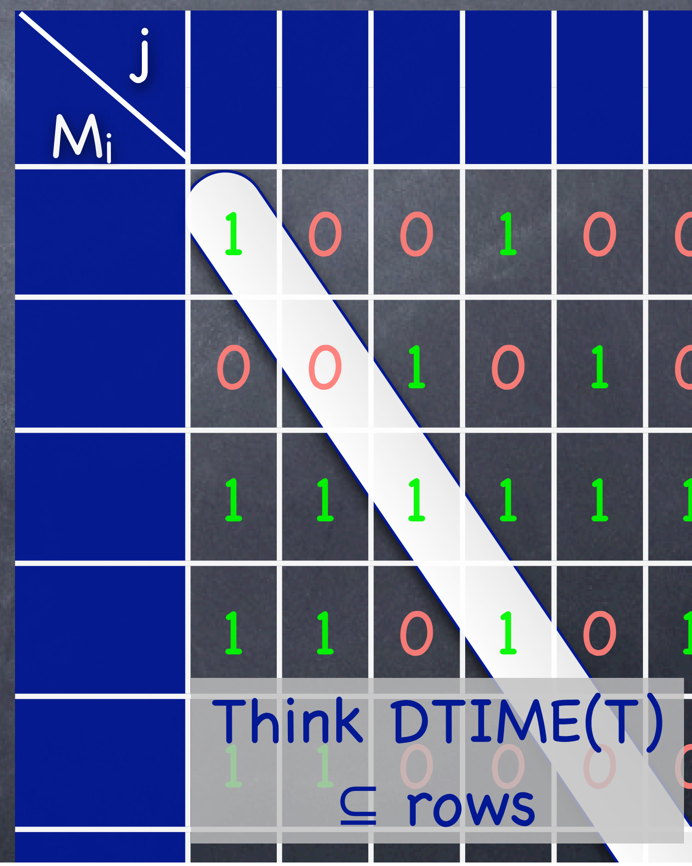


DTIME Hierarchy: Proof

- M_i be an enumeration of TMs, each TM appearing infinitely often
- Consider table(i,j) = UTM$|_{T'}$ (M_i,j), where T log T = o(T′)

T′ large and nice enough to allow simulation

- Let L′ = inverted diagonal.
- L′ in DTIME(T′)

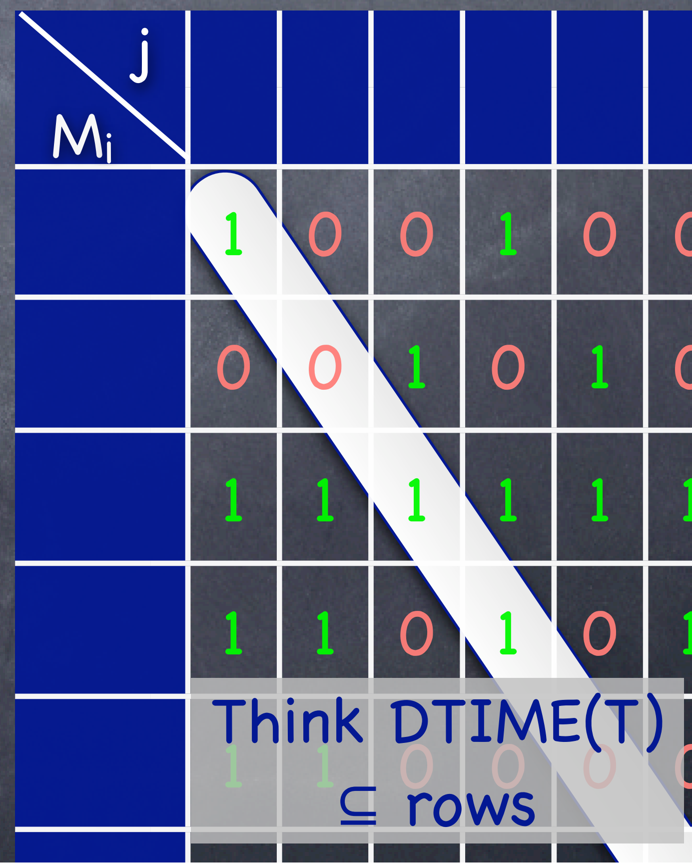


DTIME Hierarchy: Proof

- M_i be an enumeration of TMs, each TM appearing infinitely often
- Consider table(i,j) = $\mathrm{UTM}|_{T'}(M_i,j)$, where $T \log T = o(T')$

T′ large and nice enough to allow simulation

- Let L′ = inverted diagonal.
- L′ in DTIME(T′)
  - On input i, run $\mathrm{UTM}|_{T'}(M_i,i)$, modified to invert output

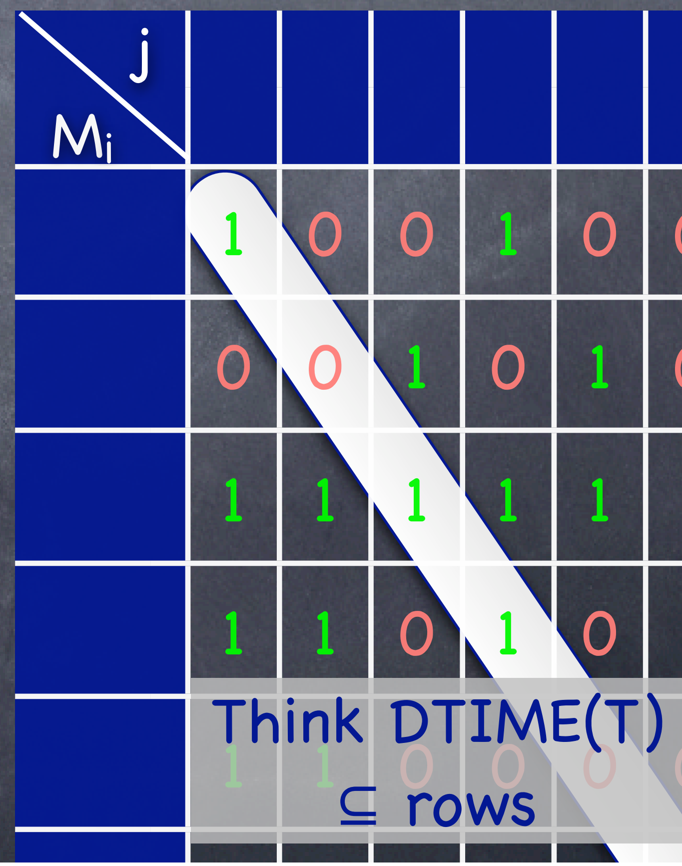


DTIME Hierarchy: Proof

- M_i be an enumeration of TMs, each TM appearing infinitely often
- Consider table(i,j) = $UTM|_{T'}$ (M_i,j), where T log T = o(T′)
- Let L′ = inverted diagonal.

T′ large and nice enough to allow simulation

- L′ in DTIME(T′)
  - On input i, run $UTM|_{T'}$ (M_i,i), modified to invert output
- L′ not in DTIME(T)

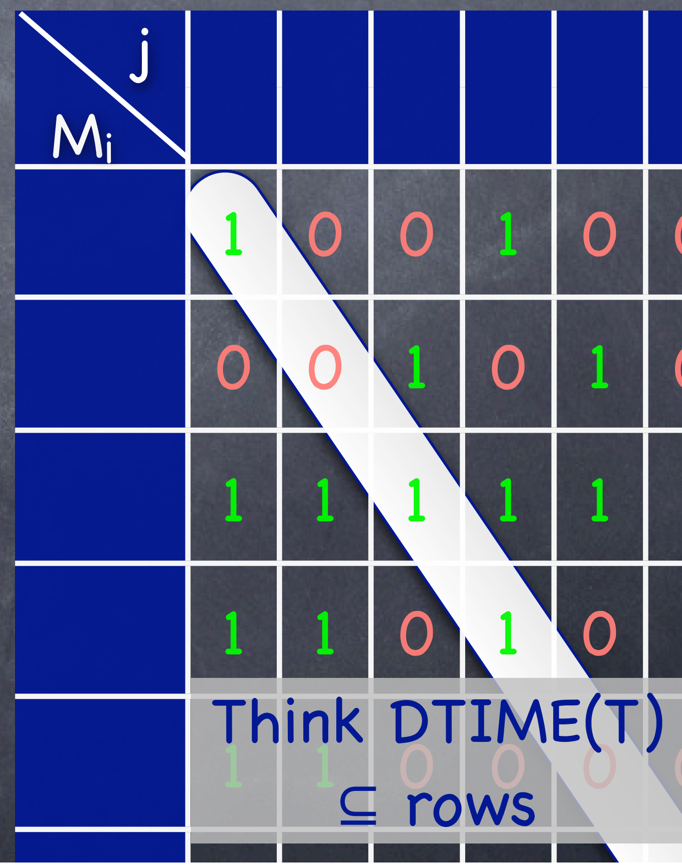


DTIME Hierarchy: Proof

- M_i be an enumeration of TMs, each TM appearing infinitely often
- Consider table(i,j) = $UTM|_{T'}$ (M_i,j), where $T \log T = o(T')$

T′ large and nice enough to allow simulation

- Let L′ = inverted diagonal.
- L′ in DTIME(T′)
    - On input i, run $UTM|_{T'}$ (M_i,i), modified to invert output
- L′ not in DTIME(T)
    - If M accepts L′ in time T, then for sufficiently large i s.t. M_i=M, UTM can finish simulating $M_i(i)$. Then table(i,i)=L′(i)!



NTIME Hierarchy

NTIME Hierarchy

- Finer hierarchy

NTIME Hierarchy

- Finer hierarchy
  - NTIME(T) $\subsetneq$ NTIME(T′) if $T(n)=o(T'(n))$, and T, T′ nice

NTIME Hierarchy

- Finer hierarchy
  - $NTIME(T) \subsetneq NTIME(T')$ if $T(n)=o(T'(n))$, and T, T' nice
  - Because a more sophisticated Universal NTM has less overhead

NTIME Hierarchy

- Finer hierarchy
  - NTIME(T) $\subsetneq$ NTIME(T') if $T(n)=o(T'(n))$, and T, T' nice
  - Because a more sophisticated Universal NTM has less overhead
- Diagonalization is more complicated

NTIME Hierarchy

- Finer hierarchy
  - NTIME(T) $\subsetneq$ NTIME(T′) if T(n)=o(T′(n)), and T, T′ nice
  - Because a more sophisticated Universal NTM has less overhead
- Diagonalization is more complicated
  - Issue: NTIME(T′) enough to simulate NTIME(T), but not to simulate co-NTIME(T)!

NTIME Hierarchy

In fact,
$T(n+1) = o(T'(n))$

- Finer hierarchy
  - NTIME(T) $\subsetneq$ NTIME(T′) if $T(n)=o(T'(n))$, and T, T′ nice
  - Because a more sophisticated Universal NTM has less overhead
- Diagonalization is more complicated
  - Issue: NTIME(T′) enough to simulate NTIME(T), but not to simulate co-NTIME(T)!

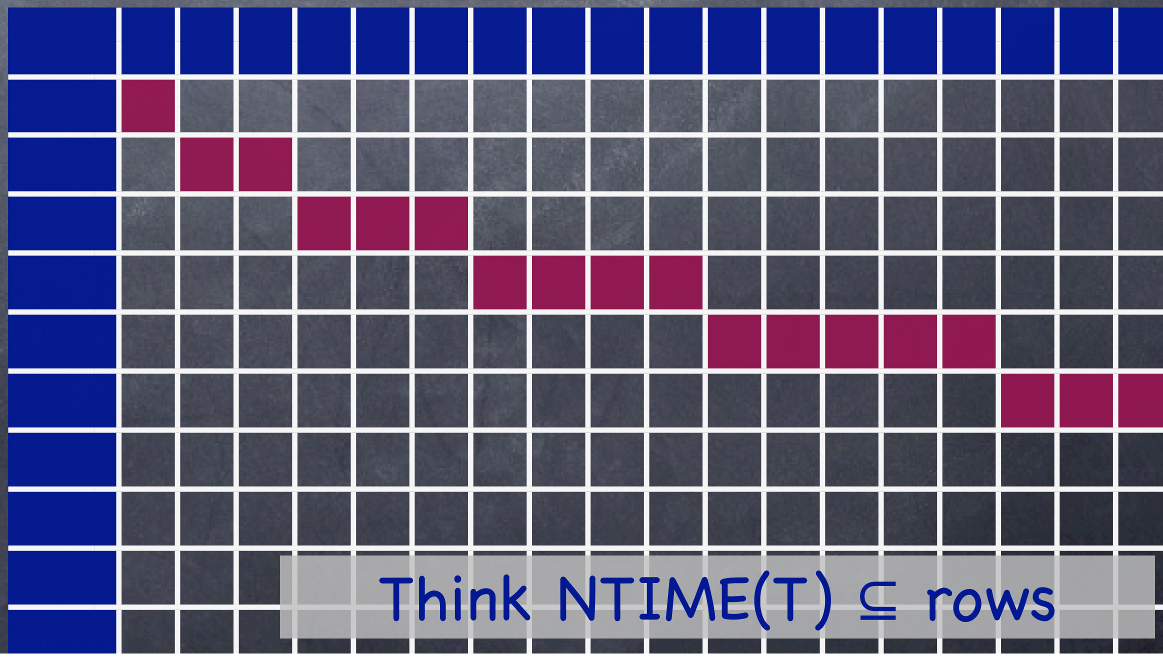
NTIME Hierarchy

NTIME Hierarchy

- “Delayed flip” on a “rapidly thickening diagonal”

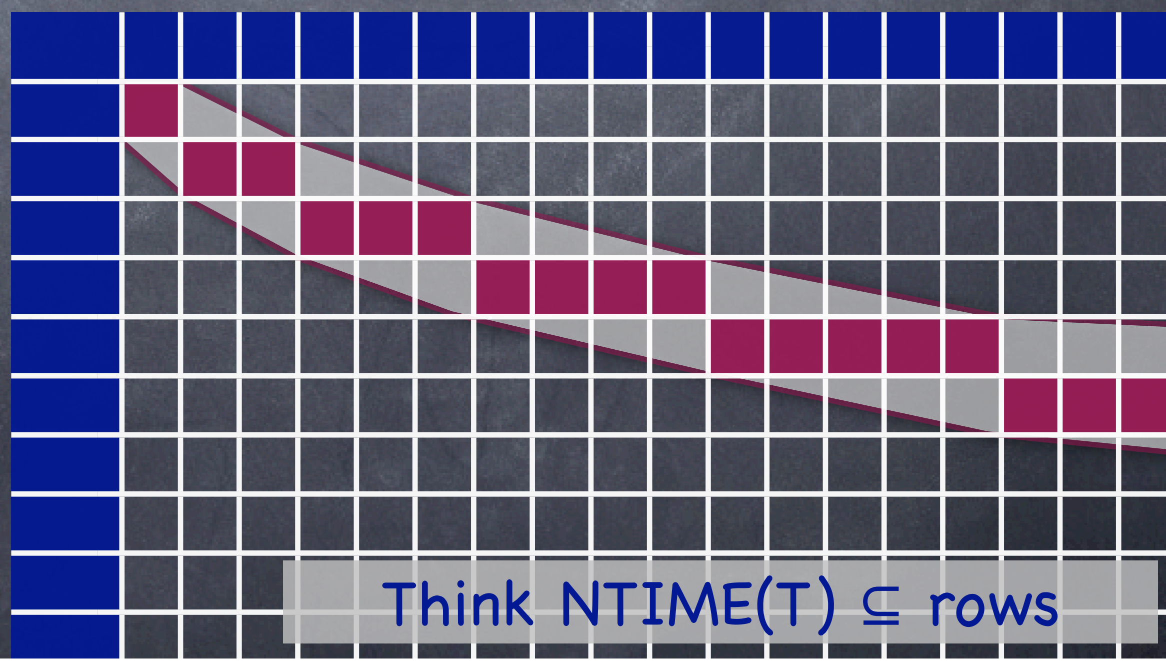
NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"



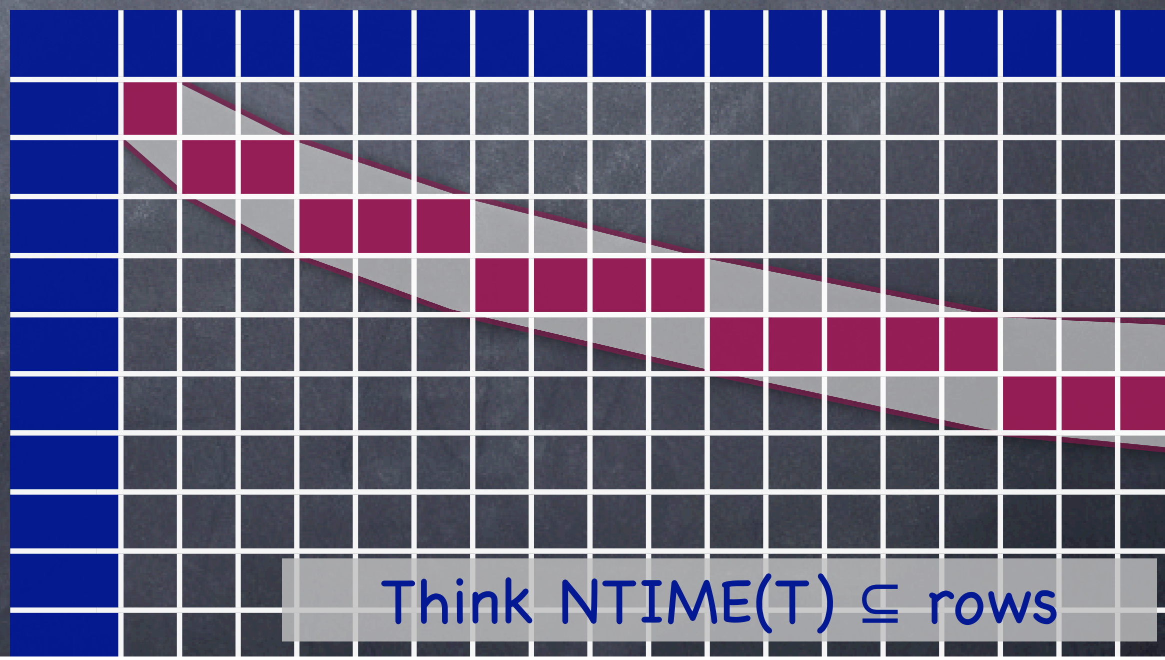
NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"



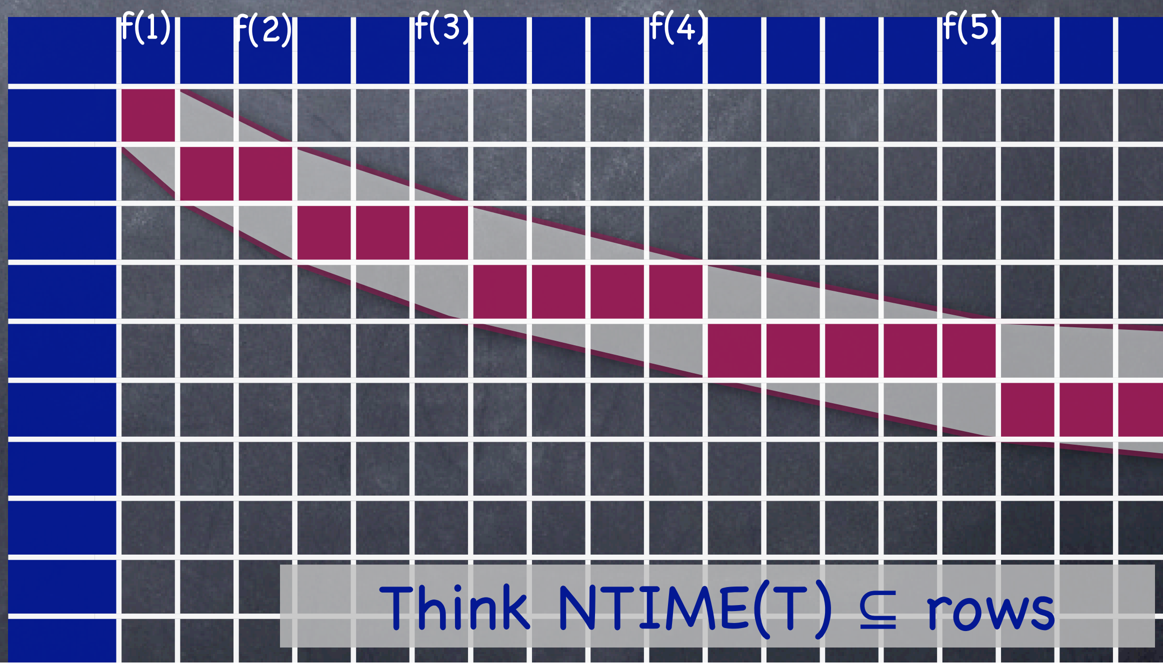
NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"
  - $f(i+1)=\exp(f(i))$



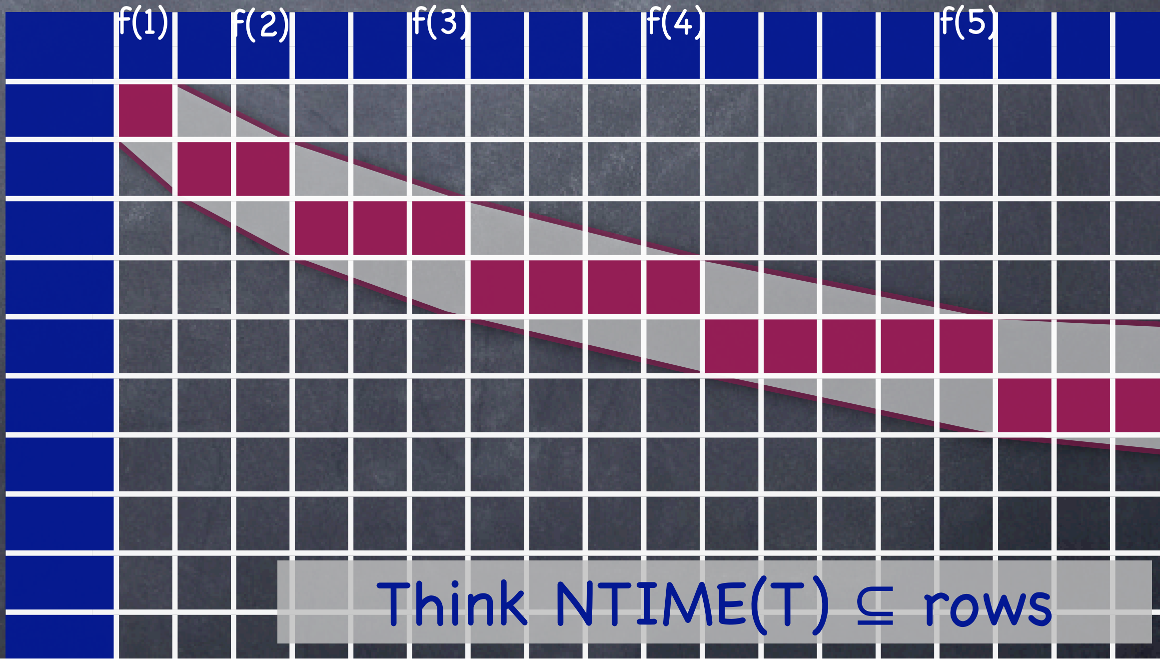
NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"
  - f(i+1)=exp(f(i))



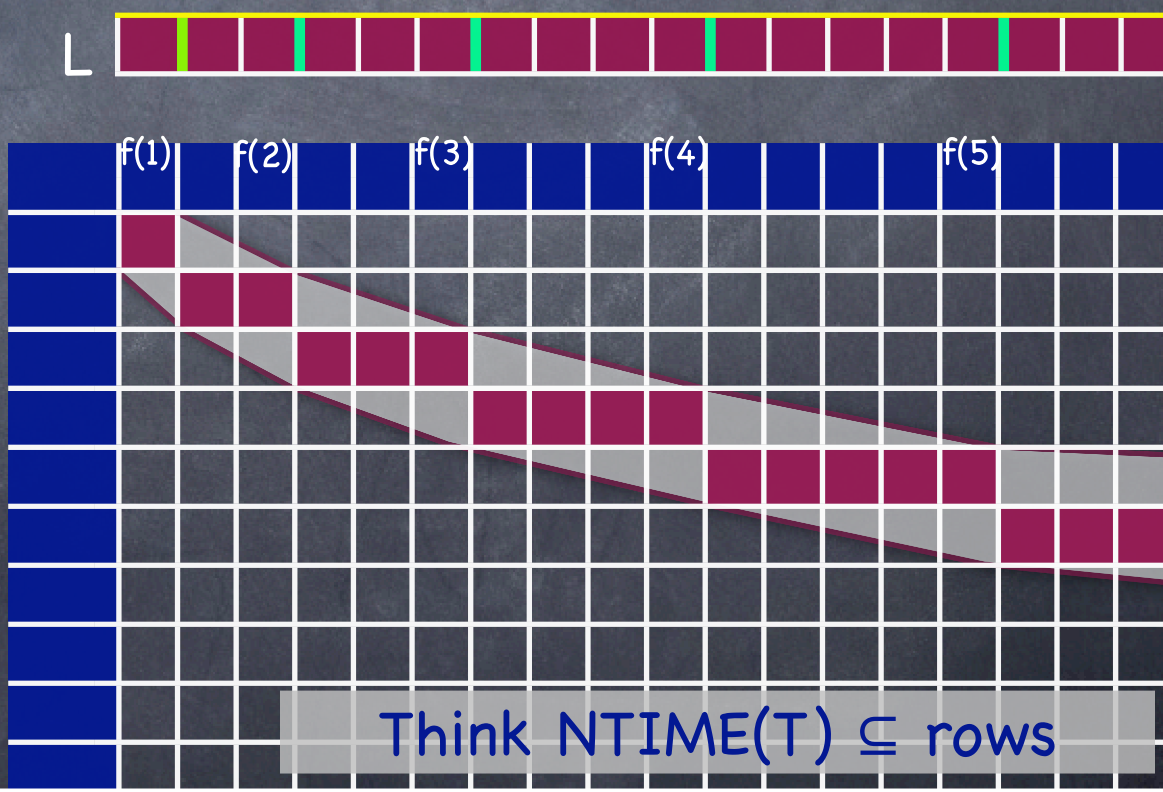
NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"
  - $f(i+1)=\exp(f(i))$
  - Let L be the "diagonal" language



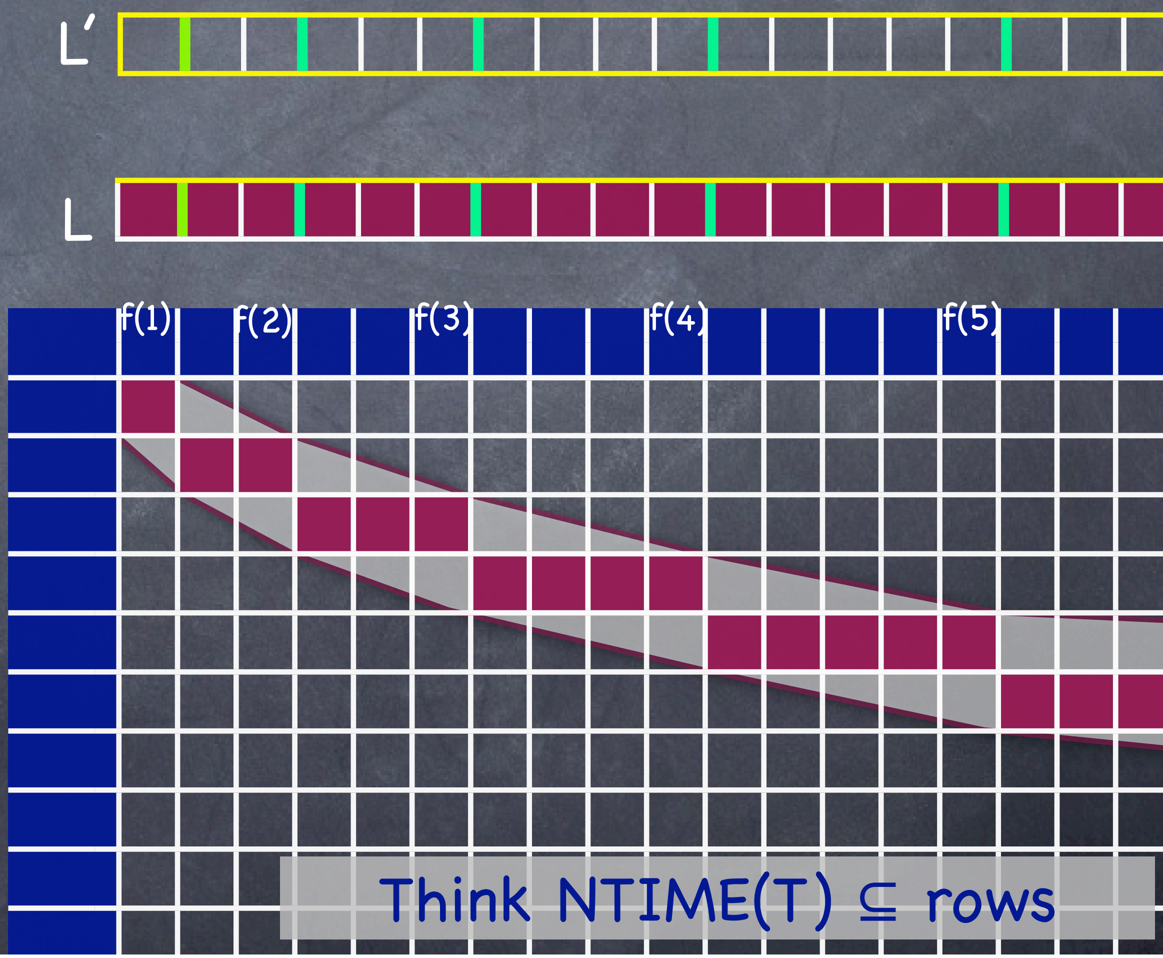
NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"
    - f(i+1)=exp(f(i))
    - Let L be the "diagonal" language



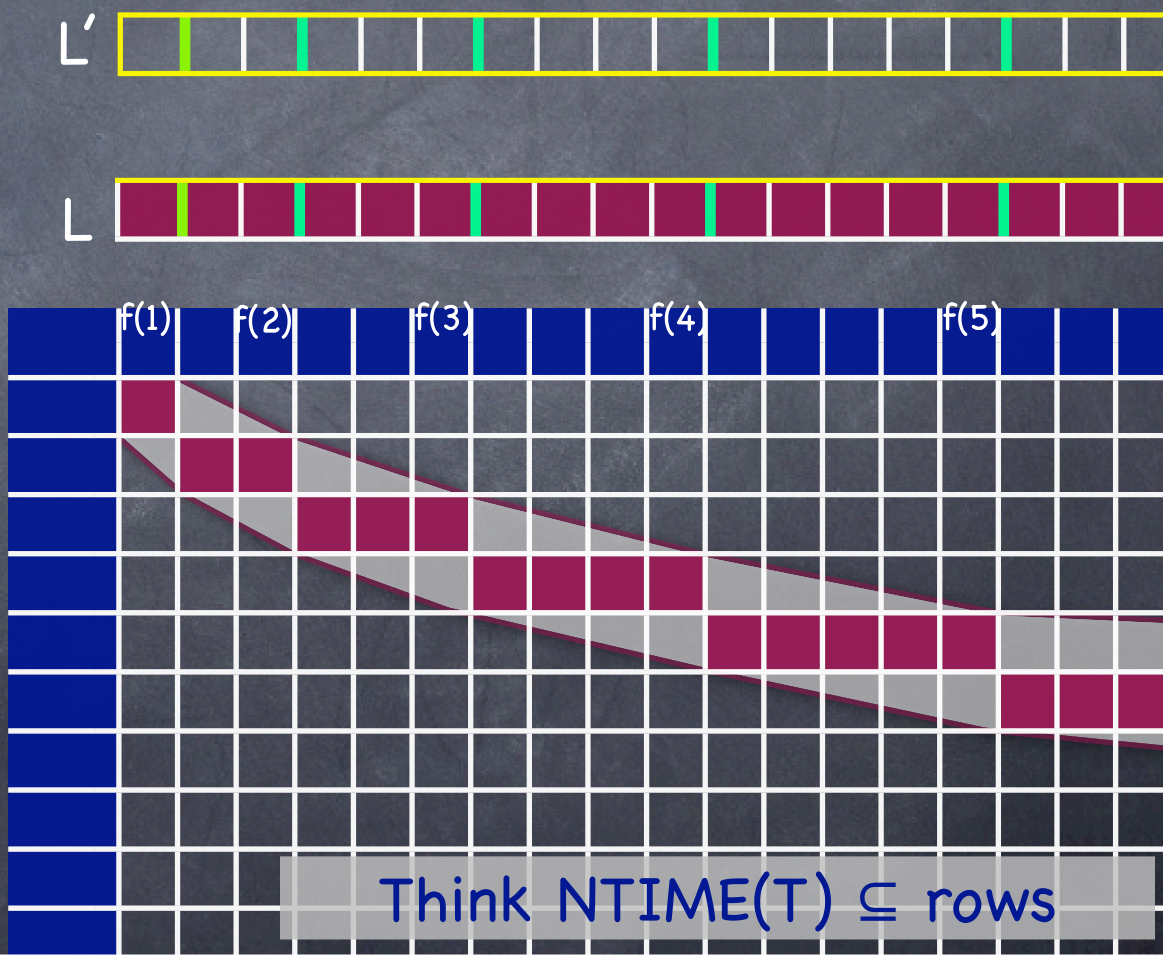
NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"
    - f(i+1)=exp(f(i))
    - Let L be the "diagonal" language



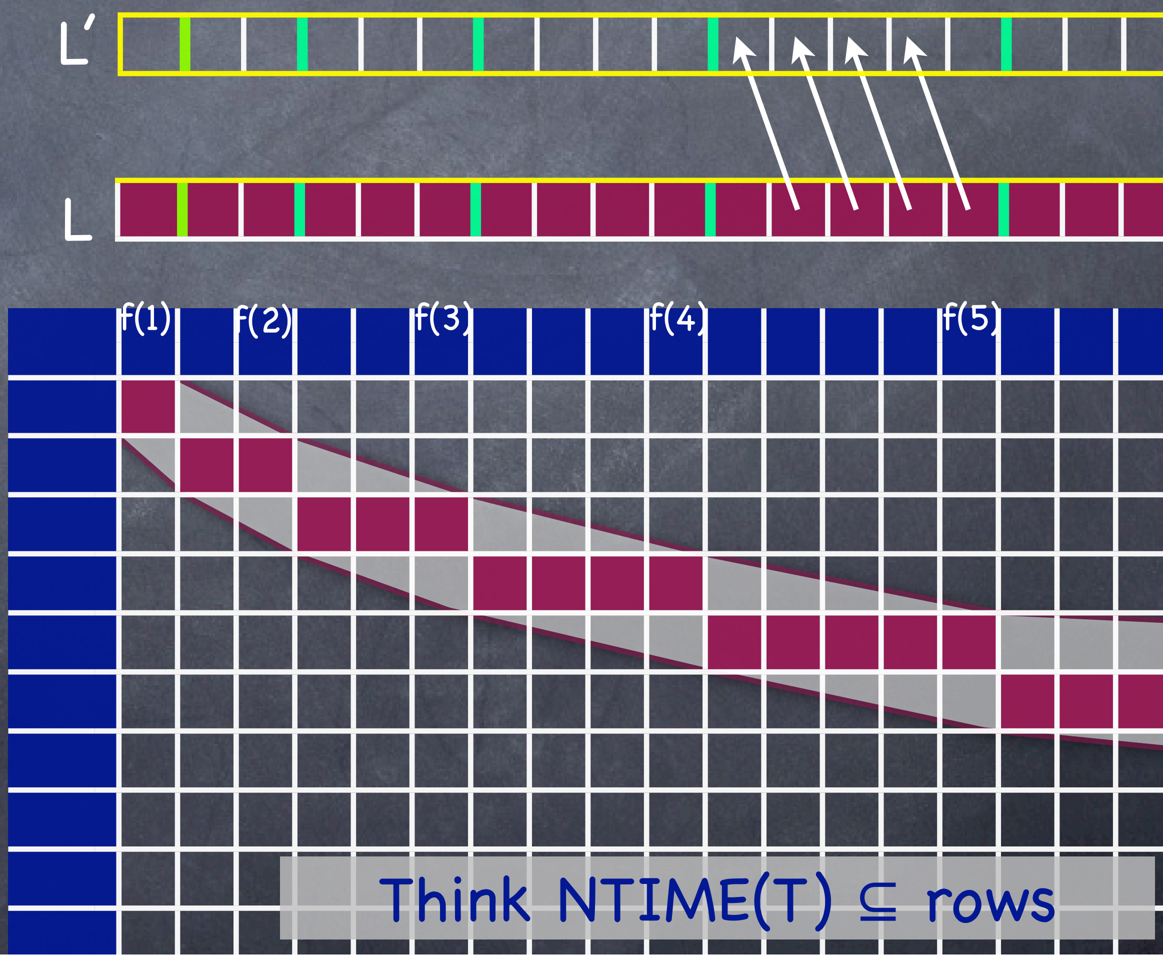
NTIME Hierarchy

- “Delayed flip” on a “rapidly thickening diagonal”
  - f(i+1)=exp(f(i))
  - Let L be the “diagonal” language
  - L′(j)=L(j+1)



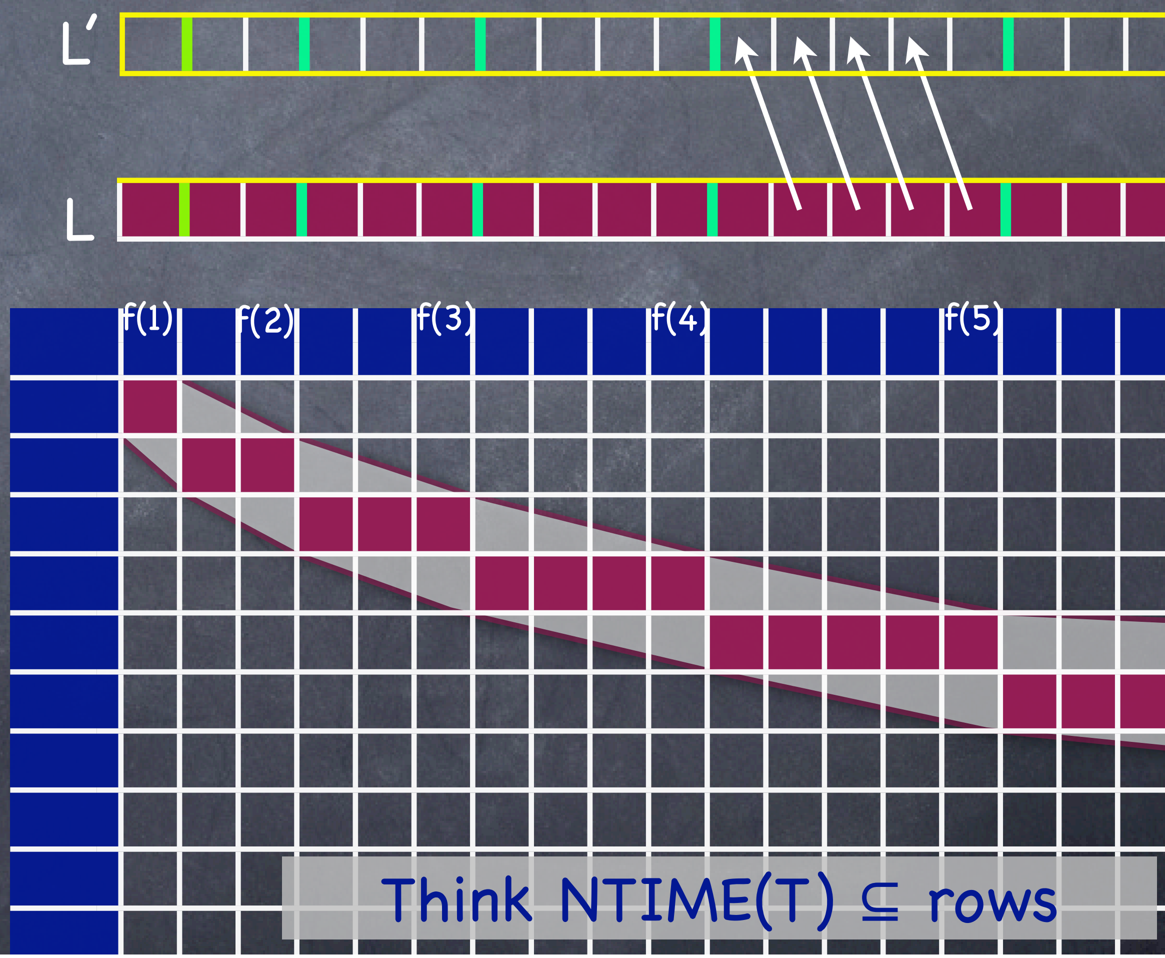
NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"
  - $f(i+1)=\exp(f(i))$
  - Let L be the "diagonal" language
  - $L'(j)=L(j+1)$



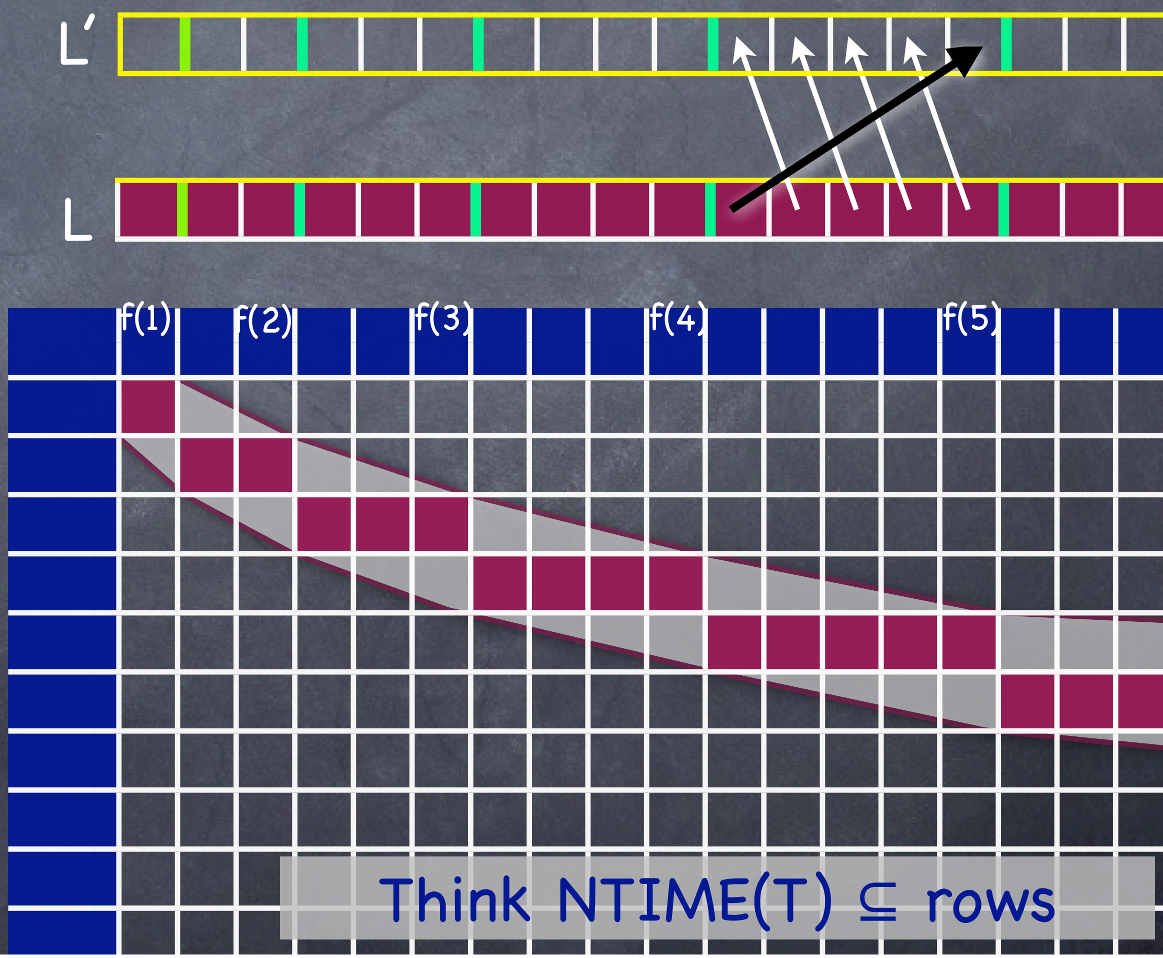
NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"
  - $f(i+1)=\exp(f(i))$
  - Let L be the "diagonal" language
  - $L'(j)=L(j+1)$
    - except if $j=f(i)$, then $L'(j) = 1 - L(f(i-1)+1)$



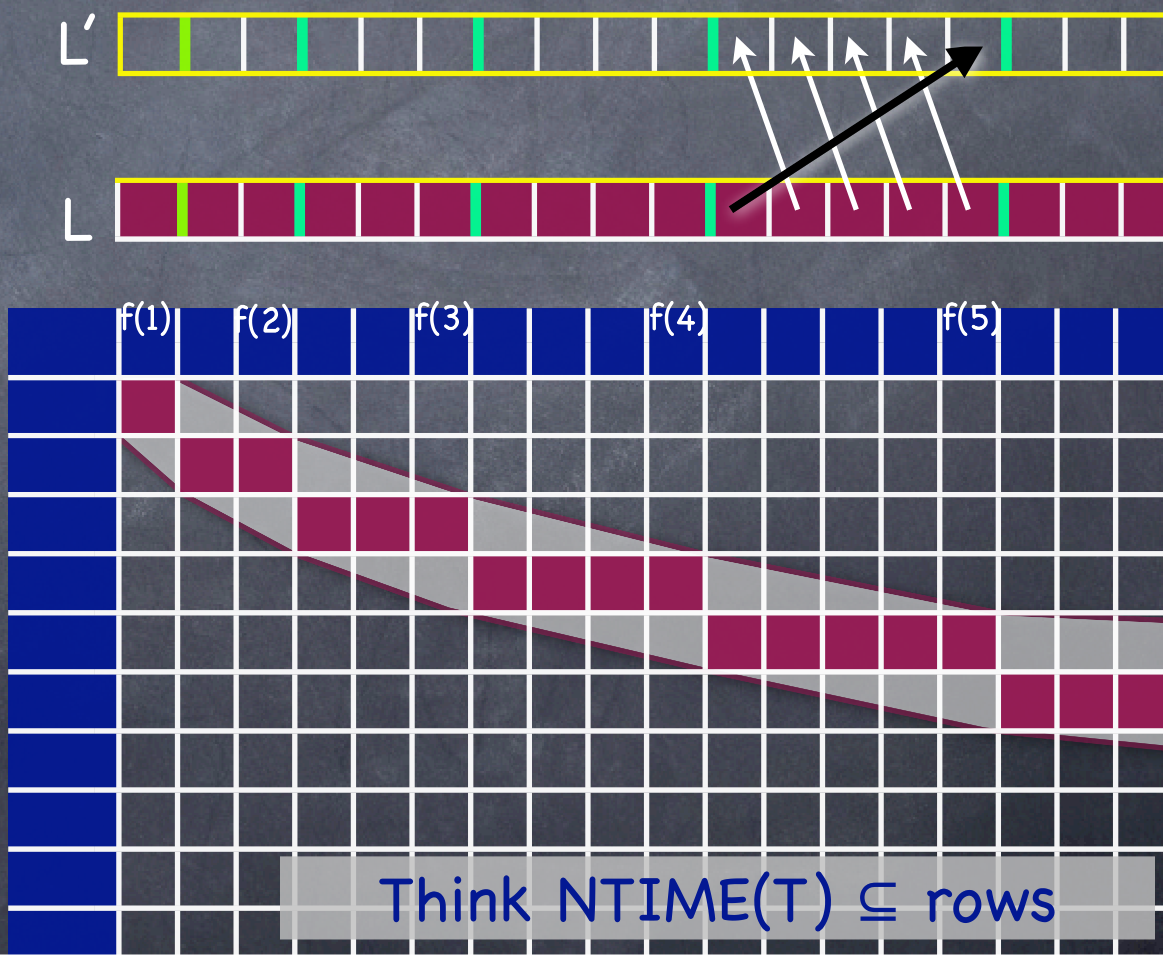
NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"
  - f(i+1)=exp(f(i))
  - Let L be the "diagonal" language
  - L′(j)=L(j+1)
    - except if j=f(i), then L′(j) = 1 - L(f(i-1)+1)



NTIME Hierarchy

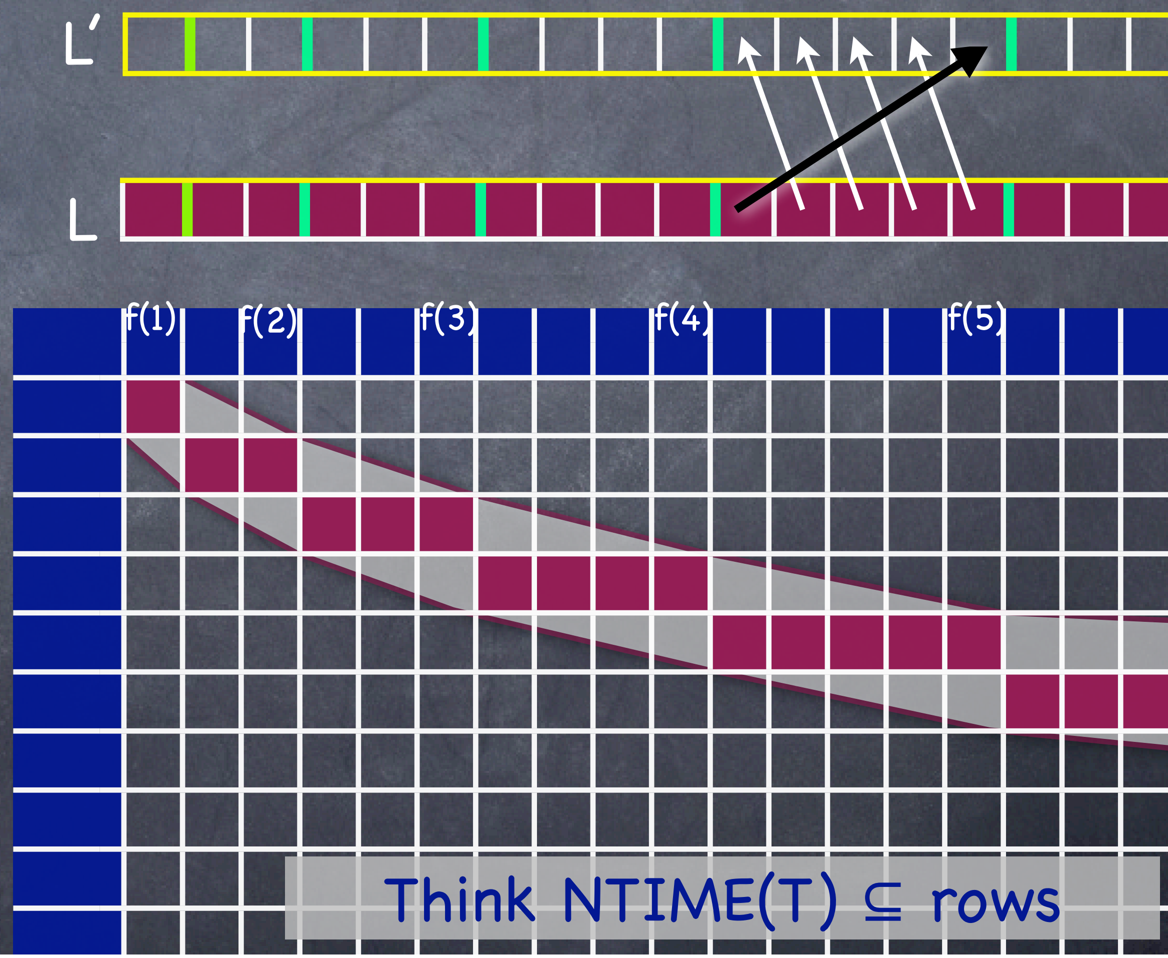
- "Delayed flip" on a "rapidly thickening diagonal"
  - $f(i+1)=\exp(f(i))$
  - Let L be the "diagonal" language
  - $L'(j)=L(j+1)$
    - except if $j=f(i)$, then $L'(j) = 1 - L(f(i-1)+1)$
  - L′ not in NTIME(T), but is in NTIME(T′)



NTIME Hierarchy

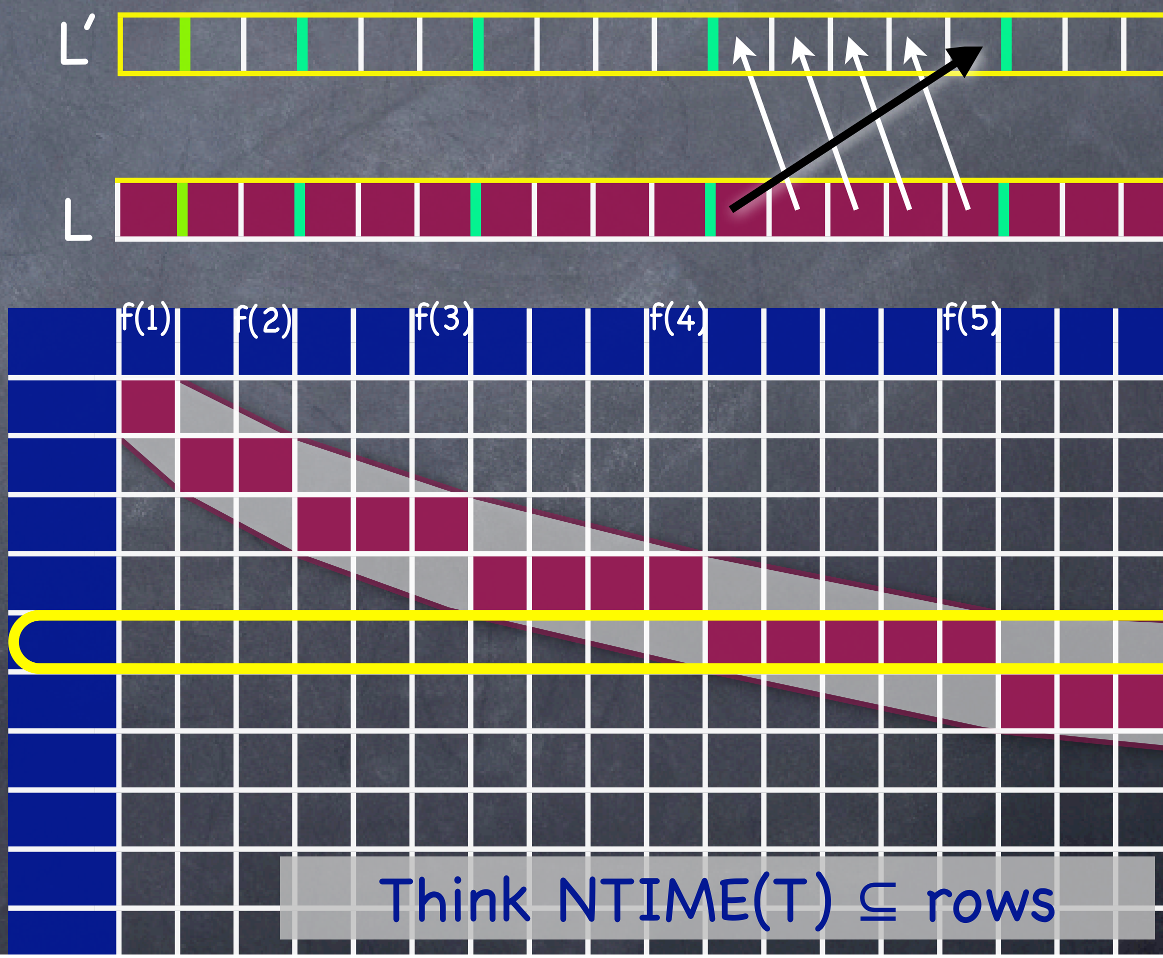
- "Delayed flip" on a "rapidly thickening diagonal"
  - f(i+1)=exp(f(i))
  - Let L be the "diagonal" language
  - L′(j)=L(j+1)
    - except if j=f(i), then L′(j) = 1 - L(f(i-1)+1)
  - L′ not in NTIME(T), but is in NTIME(T′)

Flip, Diagonal



NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"
  - $f(i+1)=\exp(f(i))$
  - Let L be the "diagonal" language
  - $L'(j)=L(j+1)$
    - except if $j=f(i)$, then $L'(j) = 1 - L(f(i-1)+1)$
  - Flip, Diagonal
  - L′ not in NTIME(T), but is in NTIME(T′)

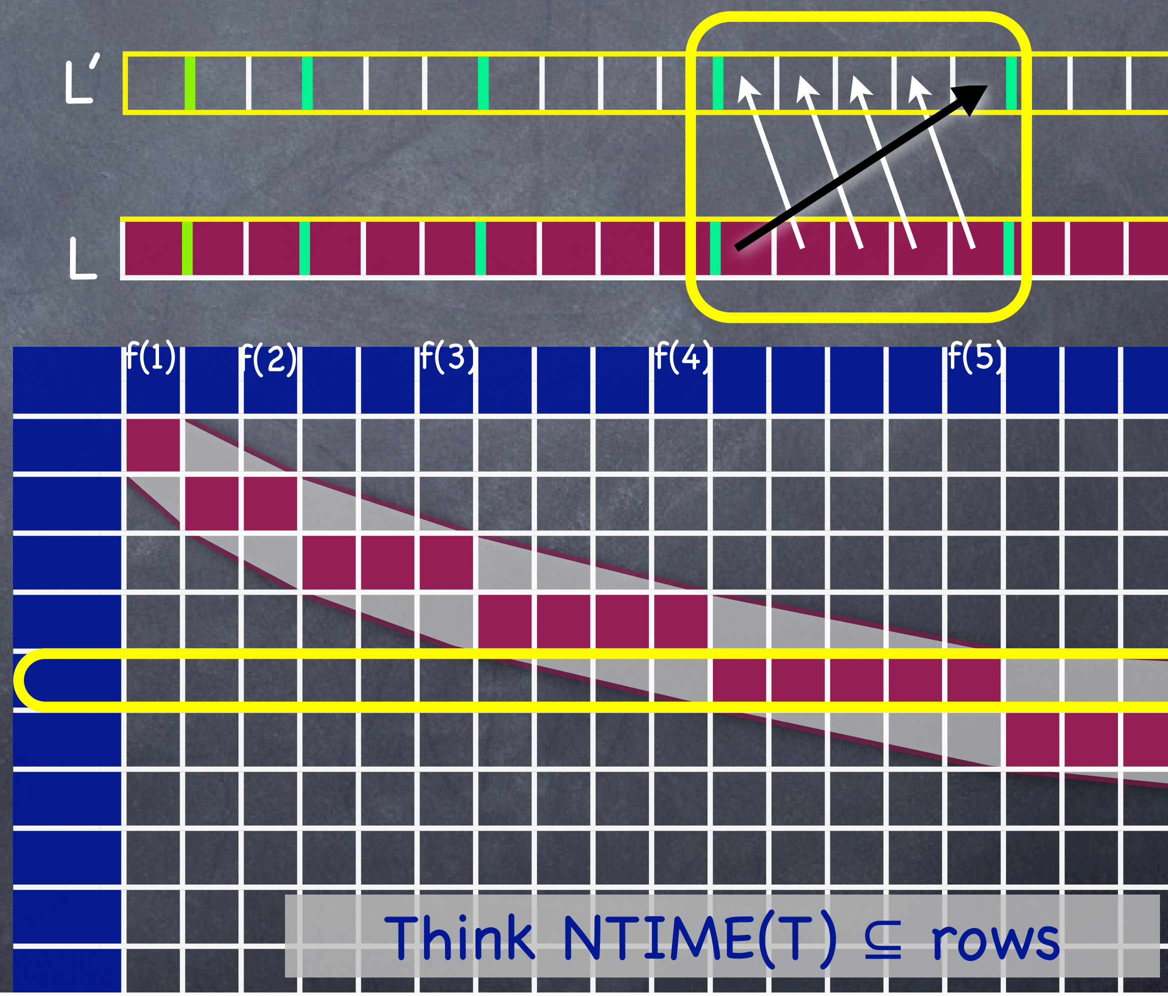


NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"
  - $f(i+1)=\exp(f(i))$
  - Let L be the "diagonal" language
  - $L'(j)=L(j+1)$
    - except if $j=f(i)$, then $L'(j) = 1 - L(f(i-1)+1)$

Flip, Diagonal

  - L′ not in NTIME(T), but is in NTIME(T′)

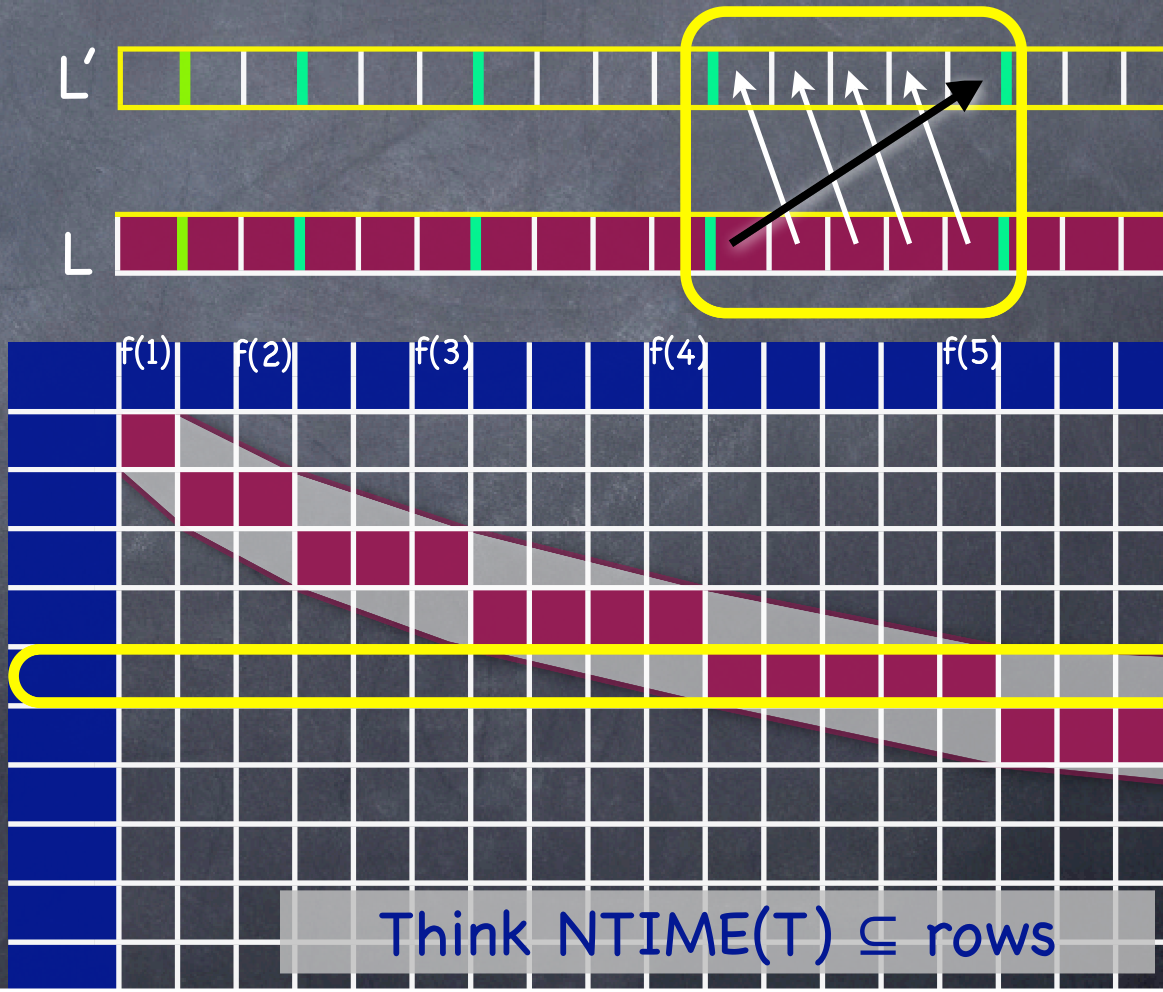


NTIME Hierarchy

- "Delayed flip" on a "rapidly thickening diagonal"
  - $f(i+1)=\exp(f(i))$
  - Let L be the "diagonal" language
  - $L'(j)=L(j+1)$
    - except if $j=f(i)$, then $L'(j) = 1 - L(f(i-1)+1)$
  - L′ not in NTIME(T), but is in NTIME(T′)

Flip, Diagonal

Delay, Rapid thickening



Time Hierarchy

Time Hierarchy

- Within DTIME and NTIME fine gradation

Time Hierarchy

- Within DTIME and NTIME fine gradation
  - In particular $P \subsetneq EXP$, $NP \subsetneq NEXP$

Time Hierarchy

- Within DTIME and NTIME fine gradation
  - In particular $P \subsetneq EXP$, $NP \subsetneq NEXP$
- Tells nothing across DTIME and NTIME

Time Hierarchy

- Within DTIME and NTIME fine gradation
  - In particular P $\subsetneq$ EXP, NP $\subsetneq$ NEXP
- Tells nothing across DTIME and NTIME
  - P and NP?

Time Hierarchy

- Within DTIME and NTIME fine gradation
  - In particular P $\subsetneq$ EXP, NP $\subsetneq$ NEXP
- Tells nothing across DTIME and NTIME
  - P and NP?
    - Just diagonalization won't help (next lecture)

Today

- DTIME Hierarchy
  - $\mathrm{DTIME}(T) \subsetneq \mathrm{DTIME}(T')$ if $T \log T = o(T')$
- NTIME Hierarchy
  - $\mathrm{NTIME}(T) \subsetneq \mathrm{NTIME}(T')$ if $T = o(T')$
- Using diagonalization

Next Lecture

- Another application of diagonalization
  - Ladner's Theorem: If P≠NP, NP language which is neither in P nor NP-complete
- Limits of Diagonalization
- Starting Space Complexity