Computational Complexity

Lecture 3
in which we come across
Diagonalization and Time-hierarchies

co-Class

co-Class

- co-X = { L | L^c is in X } (where L^c = { x | $x \notin L$ })

co-Class

- co-X = { L | L^c is in X } (where L^c = { x | $x \notin L$ })
- co-DTIME(T) = DTIME(T)

co-Class

- co-X = { L | L^c is in X } (where L^c = { x | $x \notin L$ })
- co-DTIME(T) = DTIME(T)
  - DTIME closed under complement: $L^c \in$ DTIME(T) $\Leftrightarrow$ $L \in$ DTIME(T)

co-Class

- co-X = { L | L^c is in X } (where L^c = { x | $x \notin L$ })
- co-DTIME(T) = DTIME(T)
  - DTIME closed under complement: $L^c \in$ DTIME(T) $\Leftrightarrow$ $L \in$ DTIME(T)
  - $M_{L^c} \leftrightarrow M_L$: flip accept/reject states

co-Class

- co-X = { L | L^c is in X } (where L^c = { x | $x \notin L$ })
- co-DTIME(T) = DTIME(T)
  - DTIME closed under complement: $L^c \in$ DTIME(T) $\Leftrightarrow$ $L \in$ DTIME(T)
  - $M_{L^c} \leftrightarrow M_L$: flip accept/reject states
- co-NTIME(T): all L s.t. L^c is in NTIME(T)

co-Class

- co-X = { L | L^c is in X } (where L^c = { x | $x \notin L$ })
- co-DTIME(T) = DTIME(T)
  - DTIME closed under complement: $L^c \in$ DTIME(T) $\Leftrightarrow$ $L \in$ DTIME(T)
  - $M_{L^c} \leftrightarrow M_L$: flip accept/reject states
- co-NTIME(T): all L s.t. L^c is in NTIME(T)
  - $M_{L^c} \leftrightarrow M_L$?

co-Class

- co-X = { L | L^c is in X } (where L^c = { x | $x \notin L$ })
- co-DTIME(T) = DTIME(T)
  - DTIME closed under complement: $L^c \in$ DTIME(T) $\Leftrightarrow$ $L \in$ DTIME(T)
  - $M_{L^c} \leftrightarrow M_L$: flip accept/reject states
- co-NTIME(T): all L s.t. L^c is in NTIME(T)
  - $M_{L^c} \leftrightarrow M_L$?
    - flip accept/reject states and flip "there exists" and "for all" in the acceptance criterion (NTM $\leftrightarrow$ "co-NTM")

co-Class

- co-X = { L | L^c is in X } (where L^c = { x | $x \notin L$ })
- co-DTIME(T) = DTIME(T)
  - DTIME closed under complement: $L^c \in$ DTIME(T) $\Leftrightarrow$ $L \in$ DTIME(T)
  - $M_{L^c} \leftrightarrow M_L$: flip accept/reject states
- co-NTIME(T): all L s.t. L^c is in NTIME(T)
  - $M_{L^c} \leftrightarrow M_L$?
    - flip accept/reject states and flip "there exists" and "for all" in the acceptance criterion (NTM $\leftrightarrow$ "co-NTM")
  - L^c = { x | $\nexists$w s.t. (x,w) $\in$ L' } = { x | $\forall$w (x,w) $\in$ L'^c }

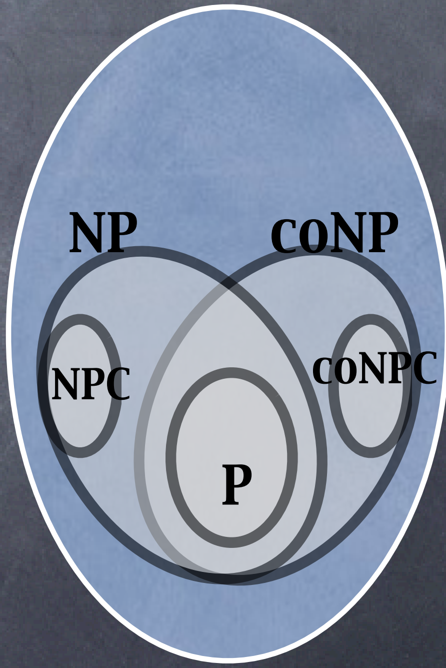
co-Class

- co-X = { L | L^c is in X } (where L^c = { x | x$\notin$L })
- co-DTIME(T) = DTIME(T)
  - DTIME closed under complement: $L^c \in$ DTIME(T) $\Leftrightarrow$ L $\in$ DTIME(T)
  - $M_{L^c} \leftrightarrow M_L$: flip accept/reject states
- co-NTIME(T): all L s.t. L^c is in NTIME(T)
  - $M_{L^c} \leftrightarrow M_L$?
    - flip accept/reject states and flip "there exists" and "for all" in the acceptance criterion (NTM $\leftrightarrow$ "co-NTM")

no counter-example

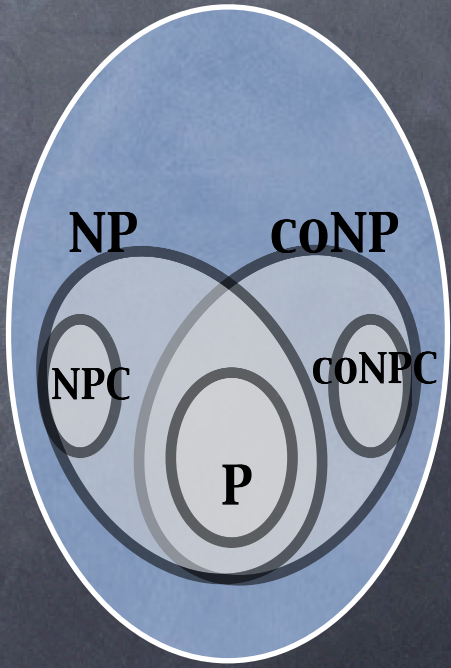
  - L^c = { x | $\nexists$w s.t. (x,w) $\in$ L' } = { x | $\forall$w (x,w) $\in$ L'^c }

NP, P, co-NP and NPC



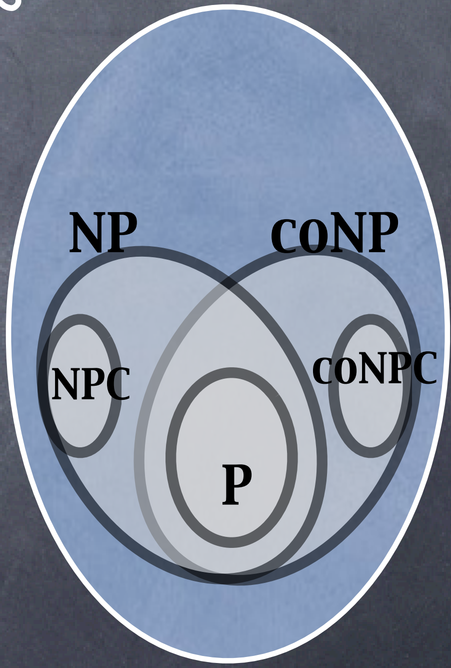
NP, P, co-NP and NPC

- 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



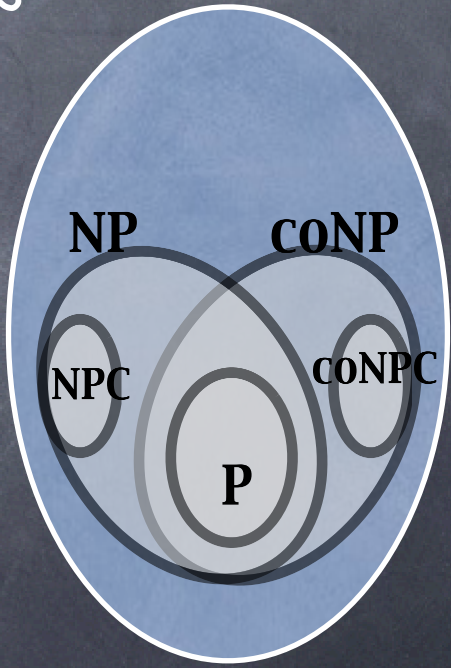
NP, P, co-NP and NPC

- 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



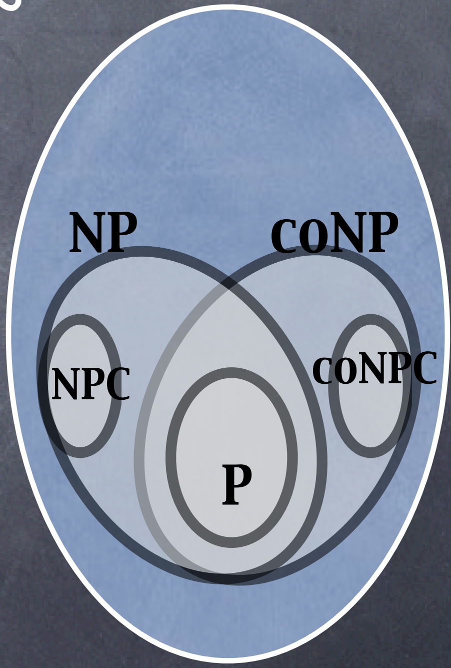
NP, P, co-NP and NPC

- 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?)



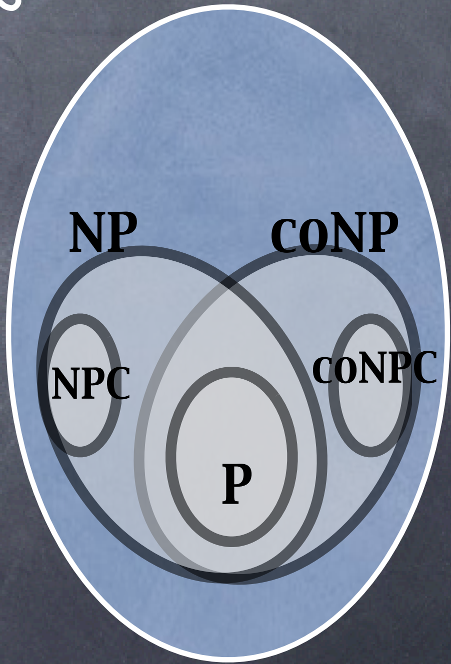
NP, P, co-NP and NPC

- 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



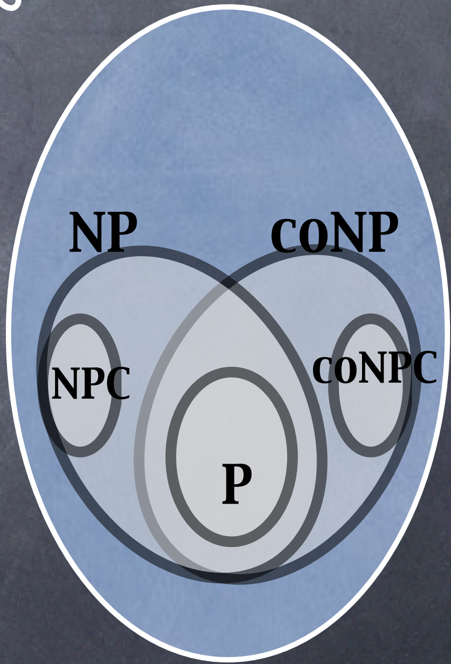
NP, P, co-NP and NPC

- 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



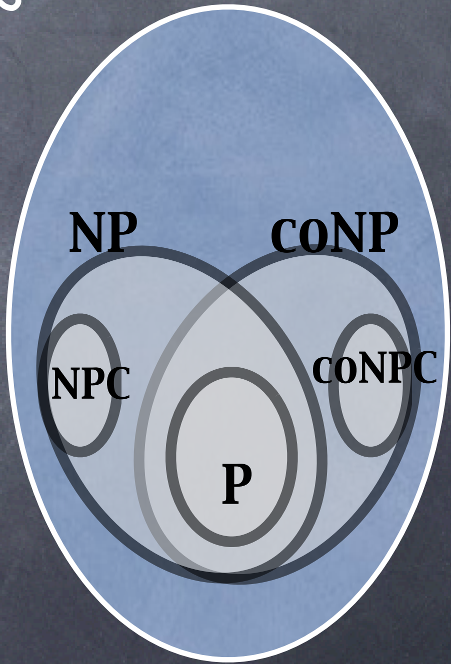
NP, P, co-NP and NPC

- 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: $X \subseteq \text{co-}X \Rightarrow X = \text{co-}X$ (Why?)



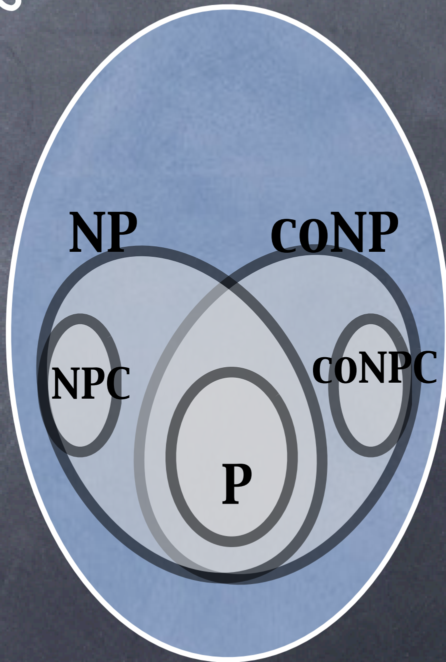
NP, P, co-NP and NPC

- 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: $X \subseteq co\text{-}X \Rightarrow X = co\text{-}X$ (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

This table can't have all 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

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: DTIME(n^c) $\subsetneq$ 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: $DTIME(n^c) \subsetneq DTIME(n^{c+1})$ for all $c \geq 1$
  - More generally $DTIME(T) \subsetneq DTIME(T')$ 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	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 $T \log T = o(T')$

Simulate up to T steps of M_i in T′ steps. 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

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 $T \log T = o(T')$

Simulate up to T steps of M_i in T′ steps. T′ large and nice enough to allow simulation

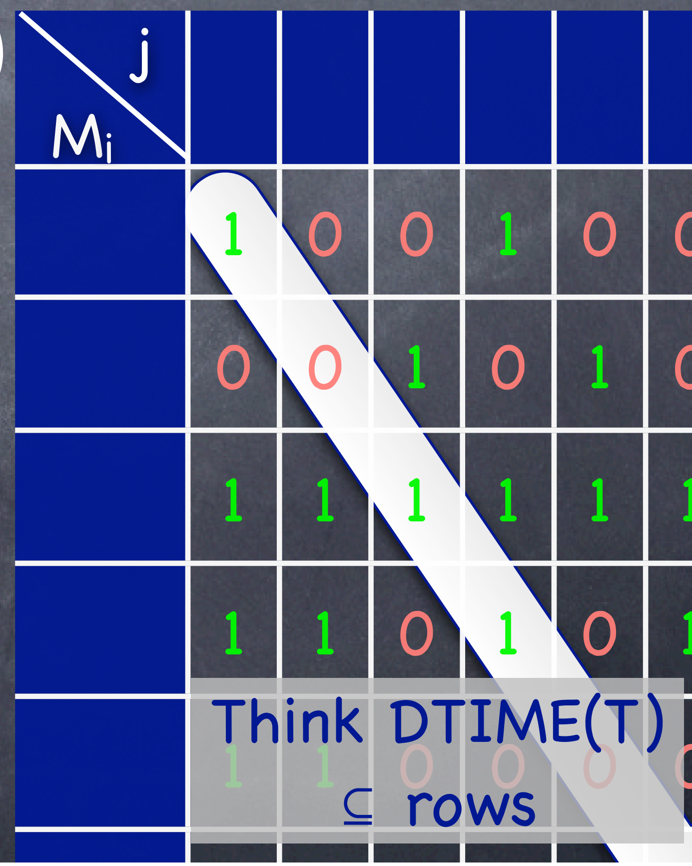
M_i \ j					
	1	0	0	1	0
	0	0	1	0	1
	1	1	1	1	1
	1	1	0	1	0
	1	1	0	0	0

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 $T \log T = o(T')$

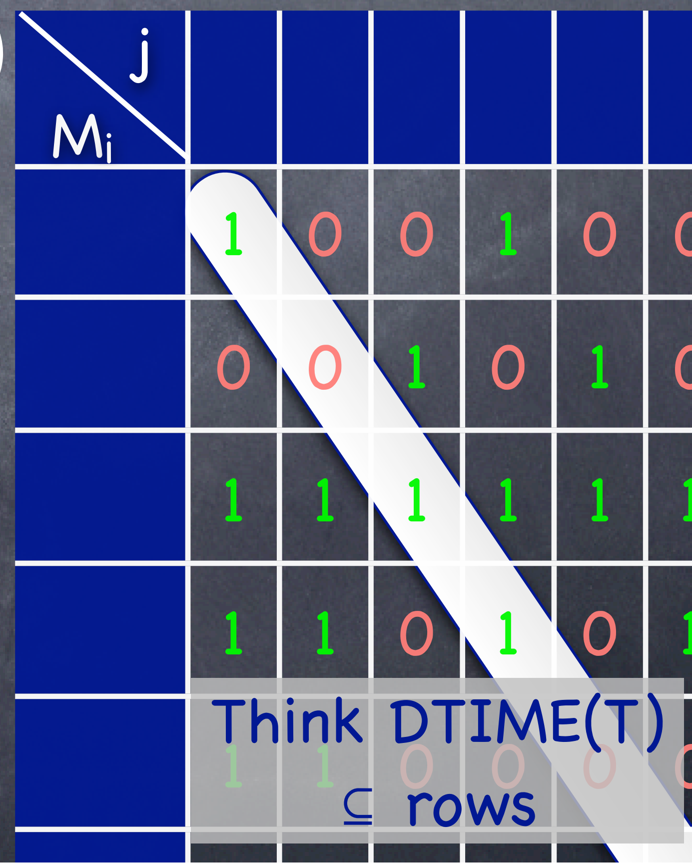
Simulate up to T steps of M_i in T′ steps. T′ large and nice enough to allow simulation



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.

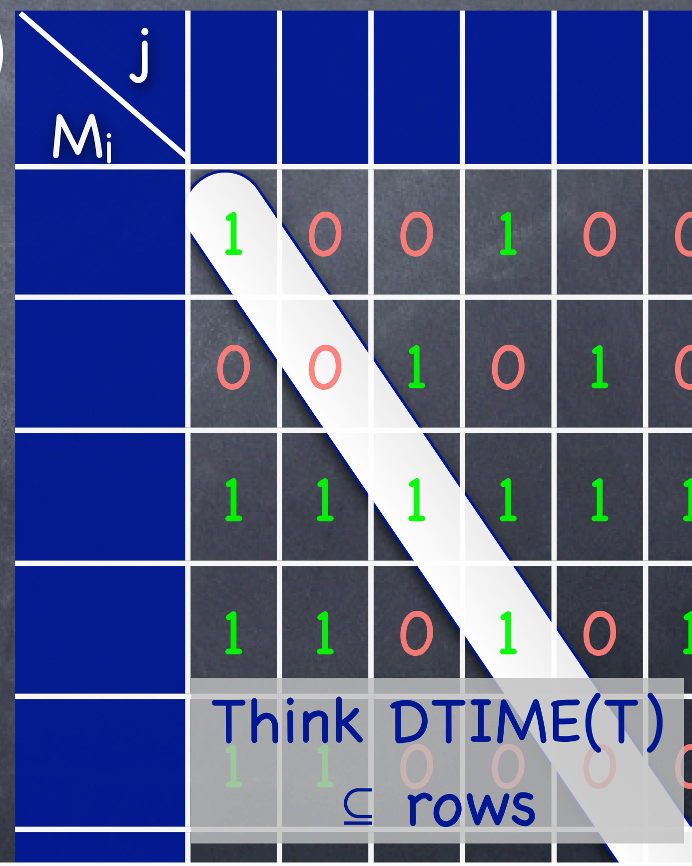
Simulate up to T steps of M_i in T′ steps. T′ large and nice enough to allow simulation



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.
- L′ in DTIME(T′)

Simulate up to T steps of M_i in T′ steps. T′ large and nice enough to allow simulation

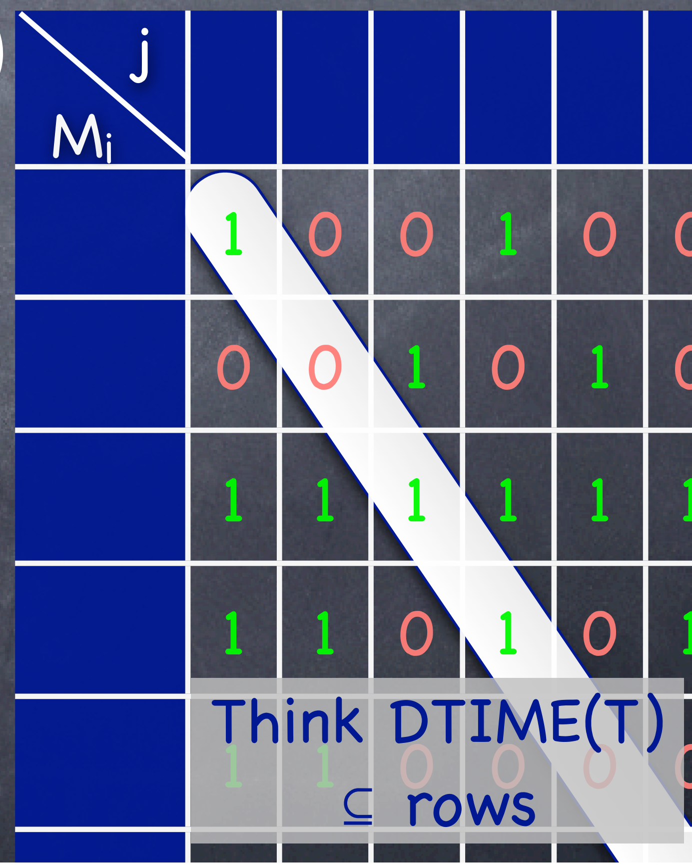


DTIME Hierarchy: Proof

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

Simulate up to T steps of M_i in T′ steps. T′ large and nice enough to allow simulation

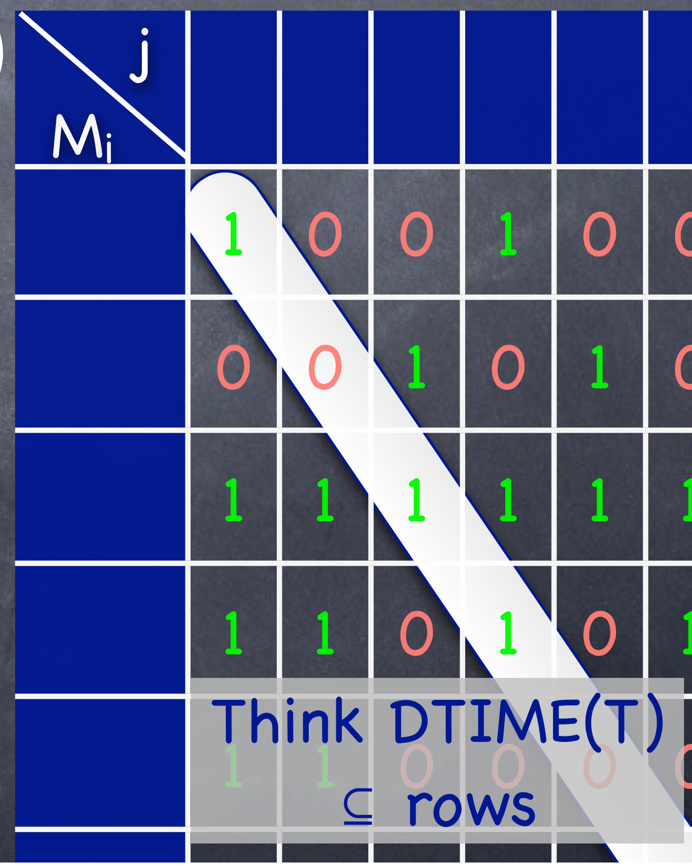
- Let L′ = inverted diagonal.
- L′ in DTIME(T′)
  - On input i, run $UTM|_{T'}$ (M_i,i), modified to invert output



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

Simulate up to T steps of M_i in T′ steps. T′ large and nice enough to allow simulation

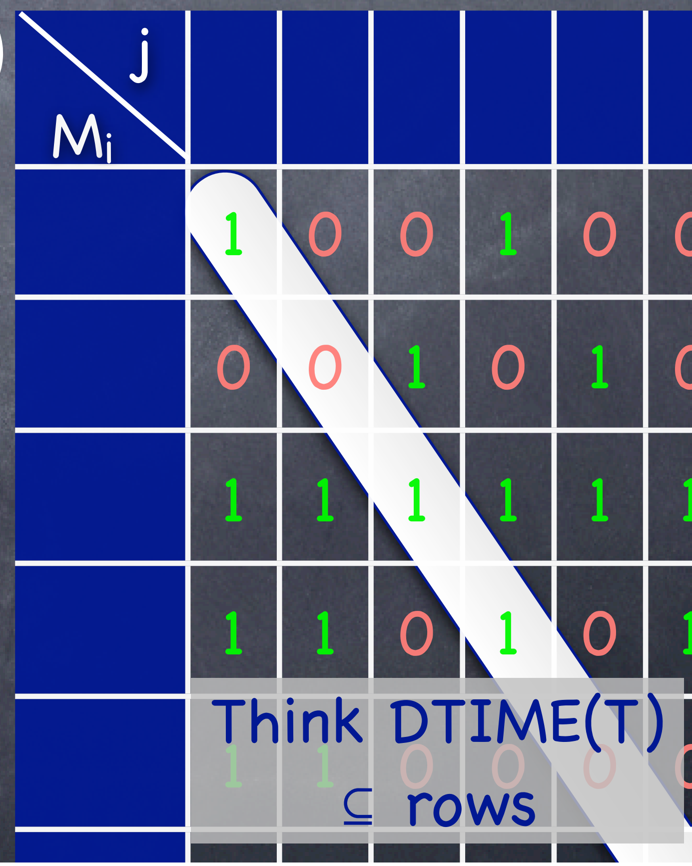


DTIME Hierarchy: Proof

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

Simulate up to T steps of M_i in T′ steps. 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

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

NTIME Hierarchy

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

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

NTIME Hierarchy

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

- Finer hierarchy
  - NTIME(T) ⊊ 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) ⊊ 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)!

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

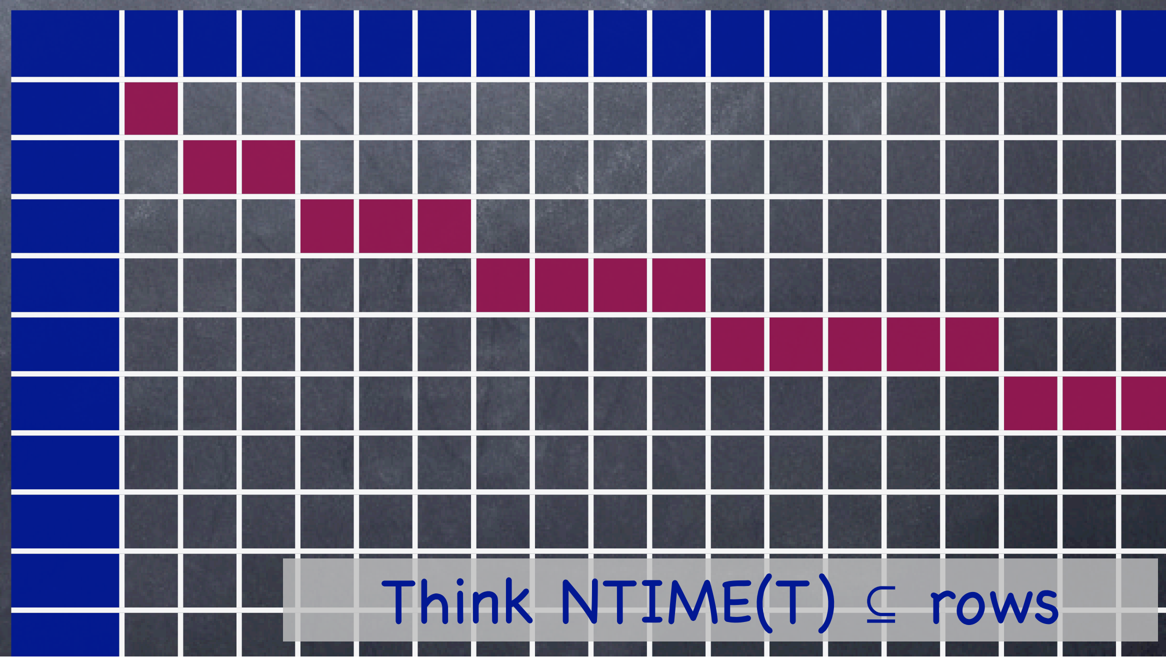
NTIME Hierarchy

NTIME Hierarchy

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

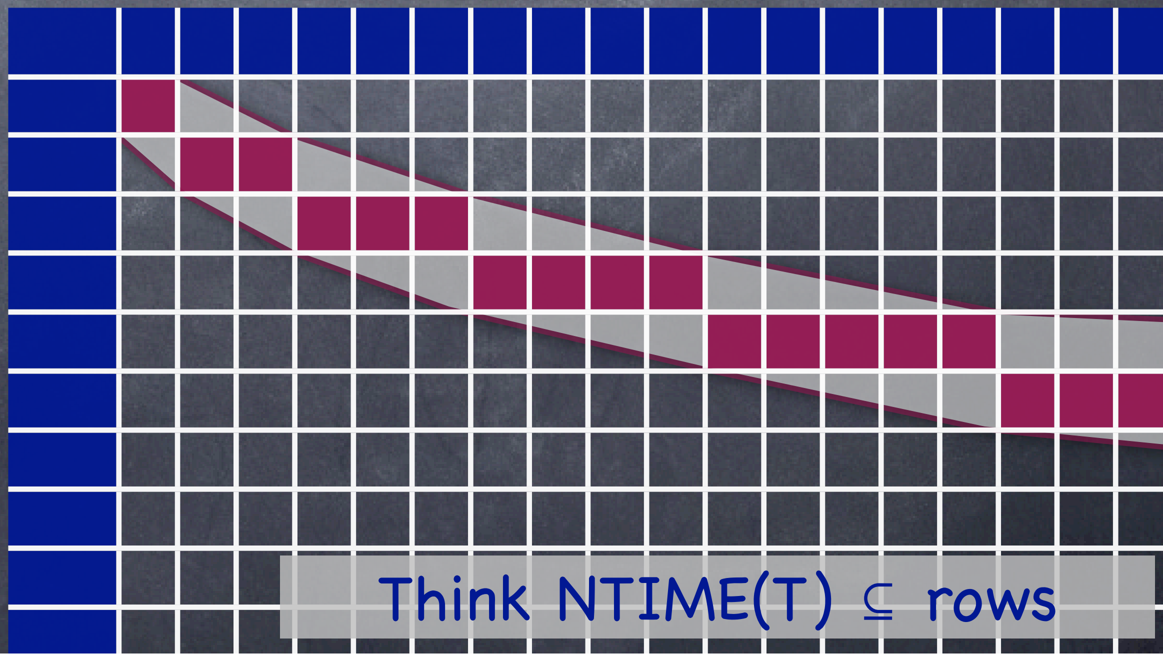
NTIME Hierarchy

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



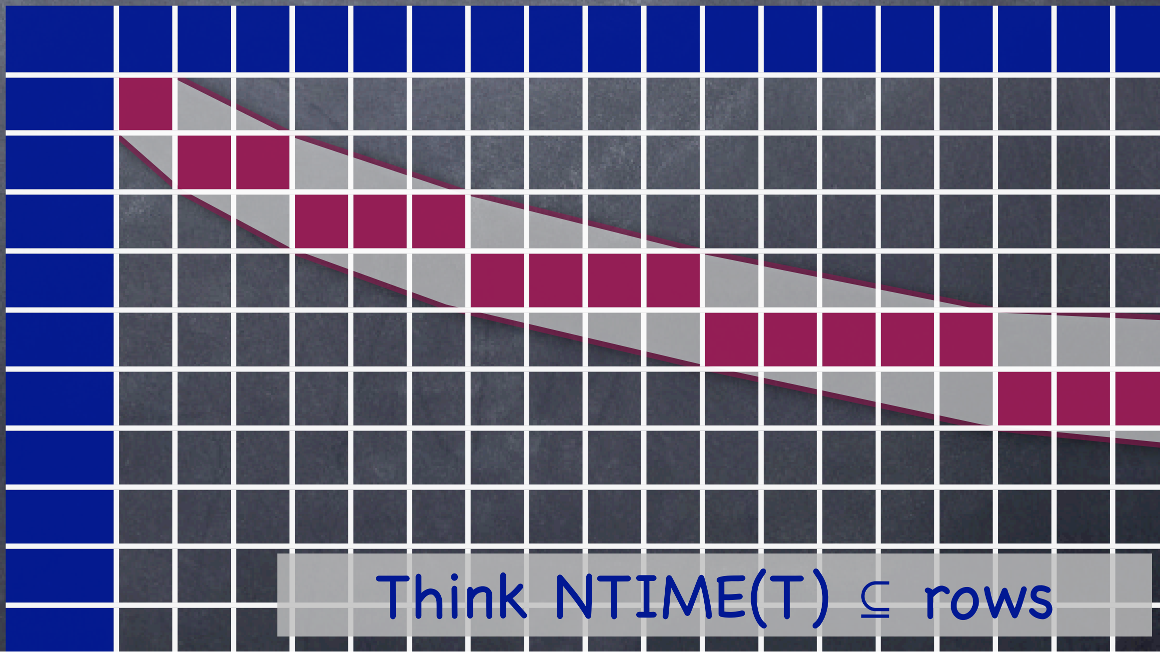
NTIME Hierarchy

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



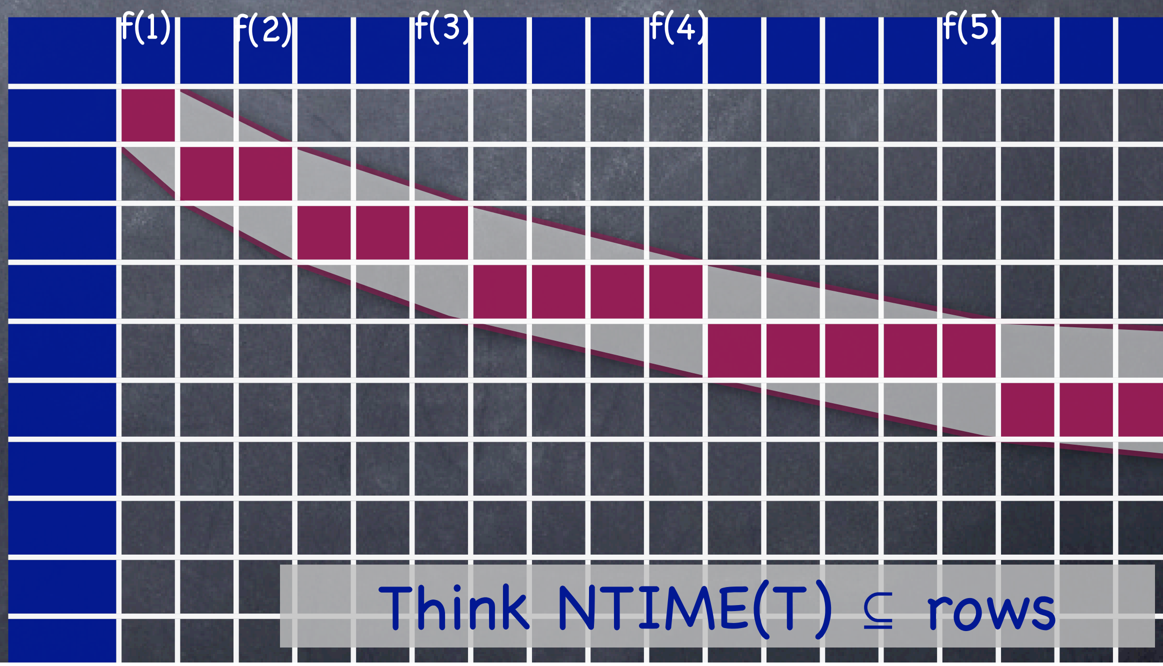
NTIME Hierarchy

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



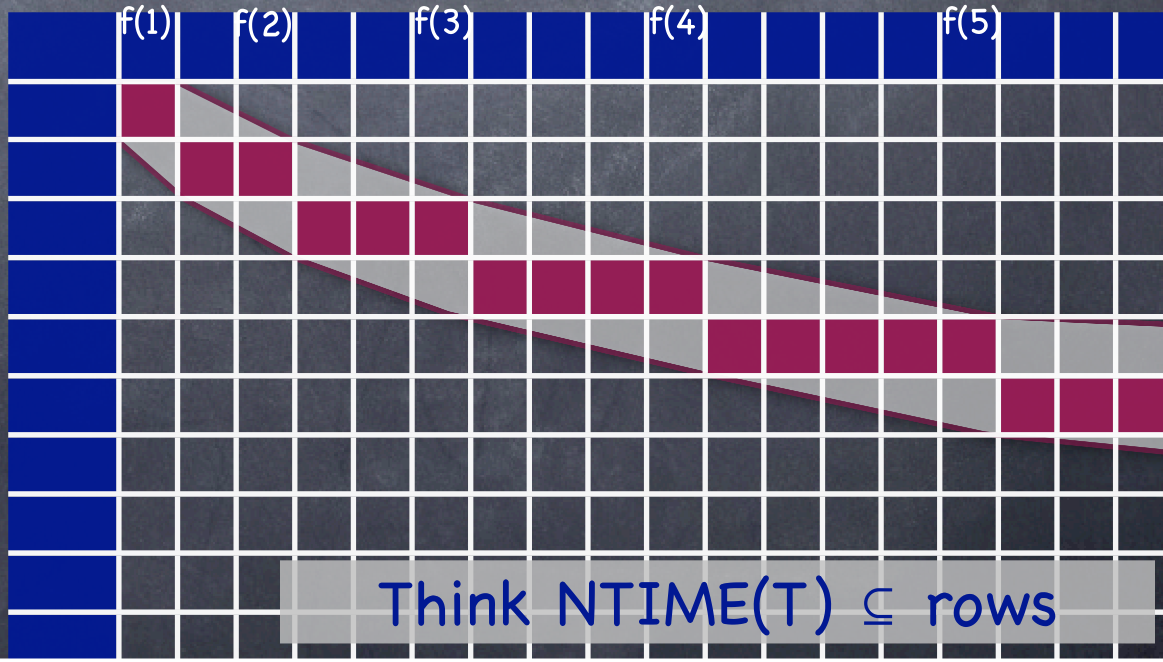
NTIME Hierarchy

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



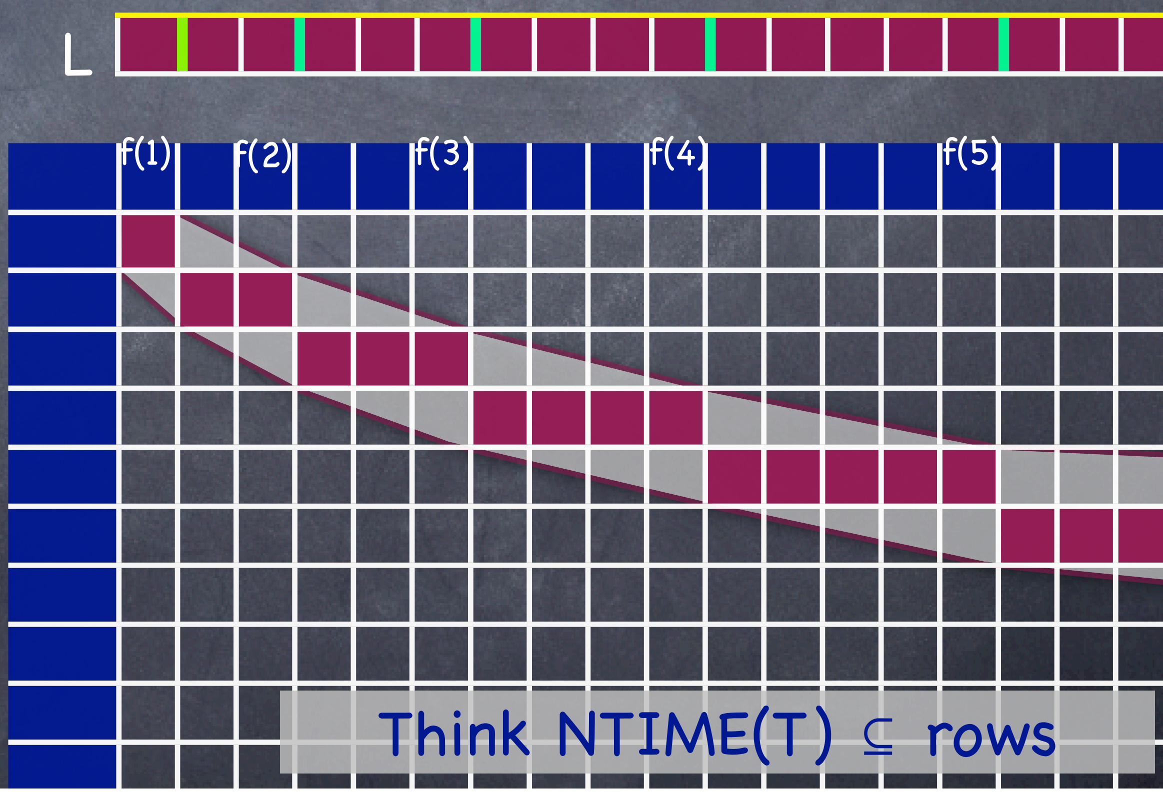
NTIME Hierarchy

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



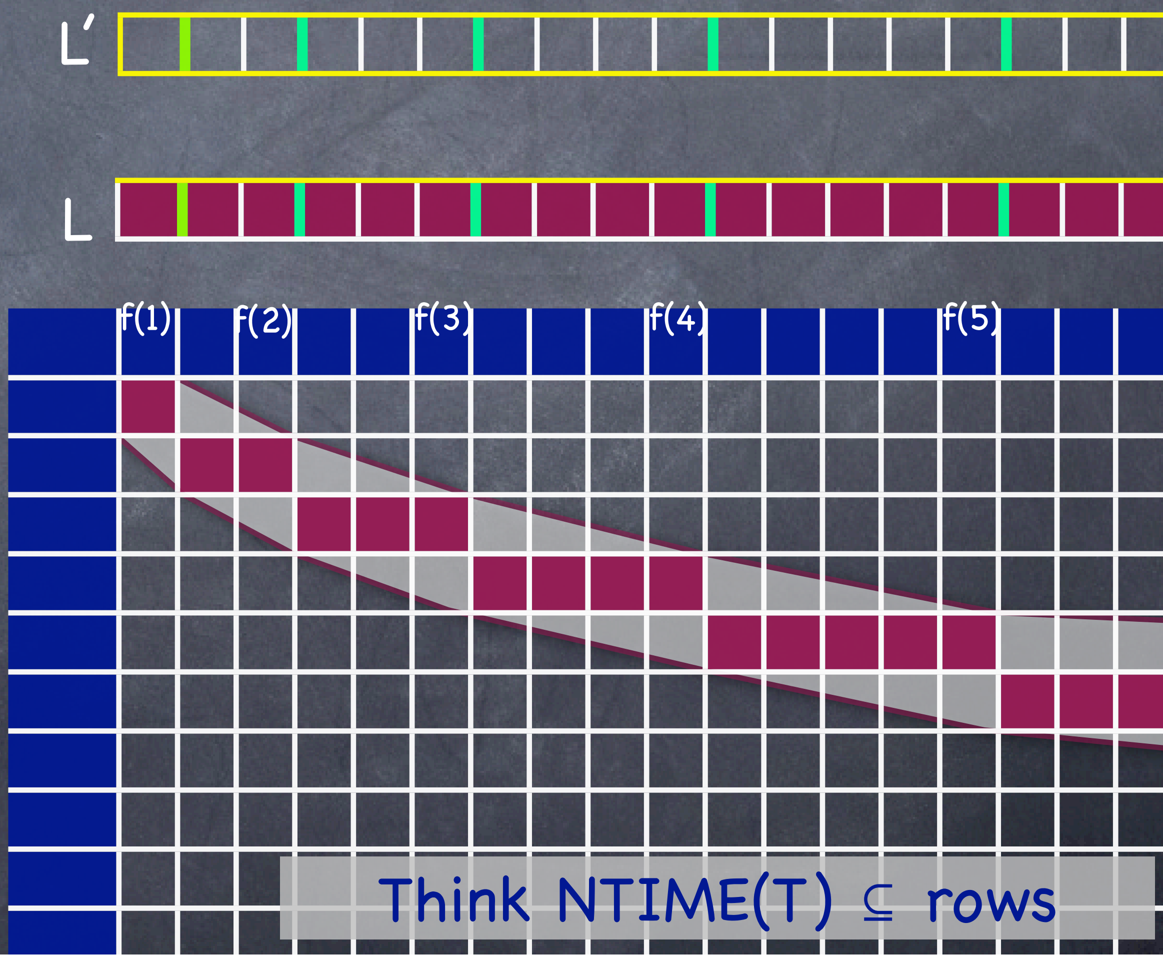
NTIME Hierarchy

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



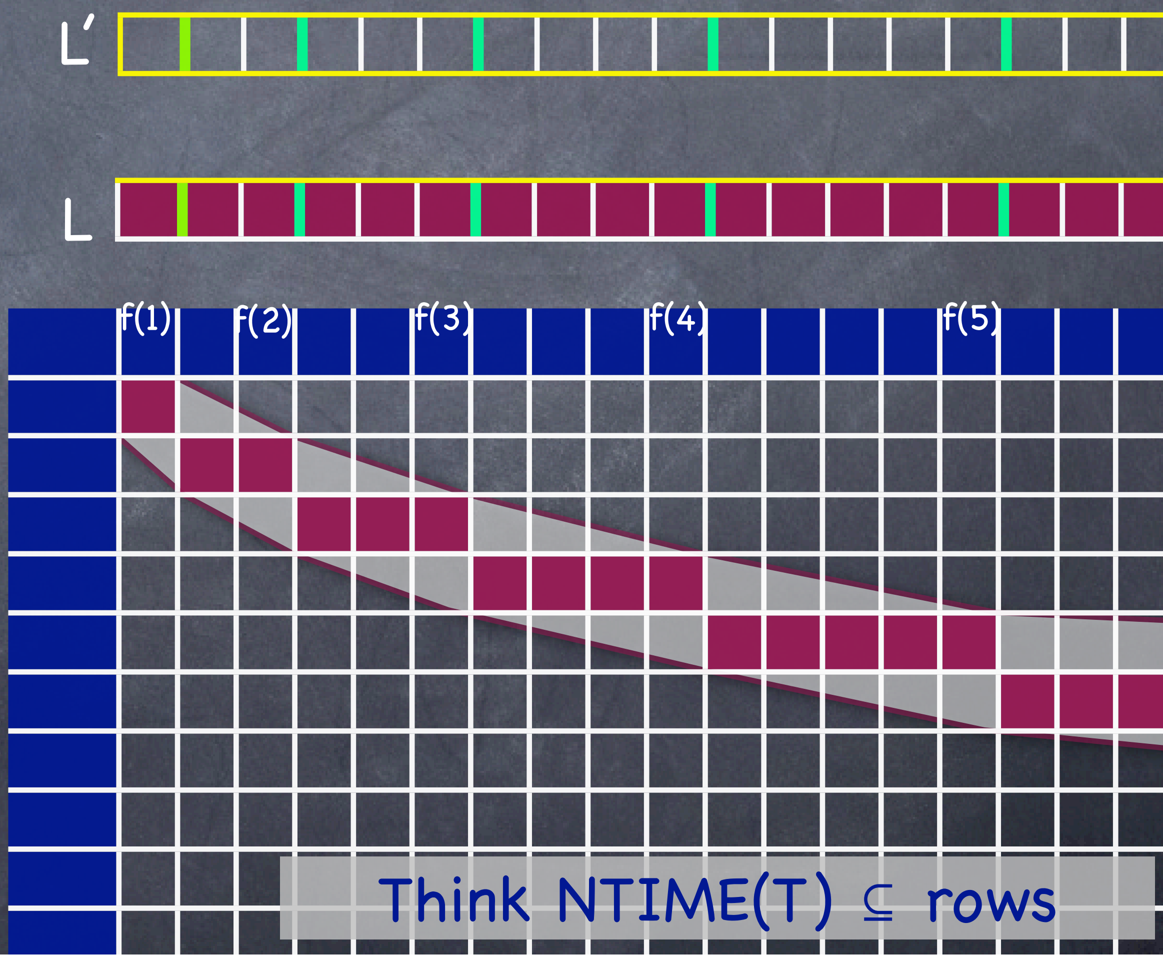
NTIME Hierarchy

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



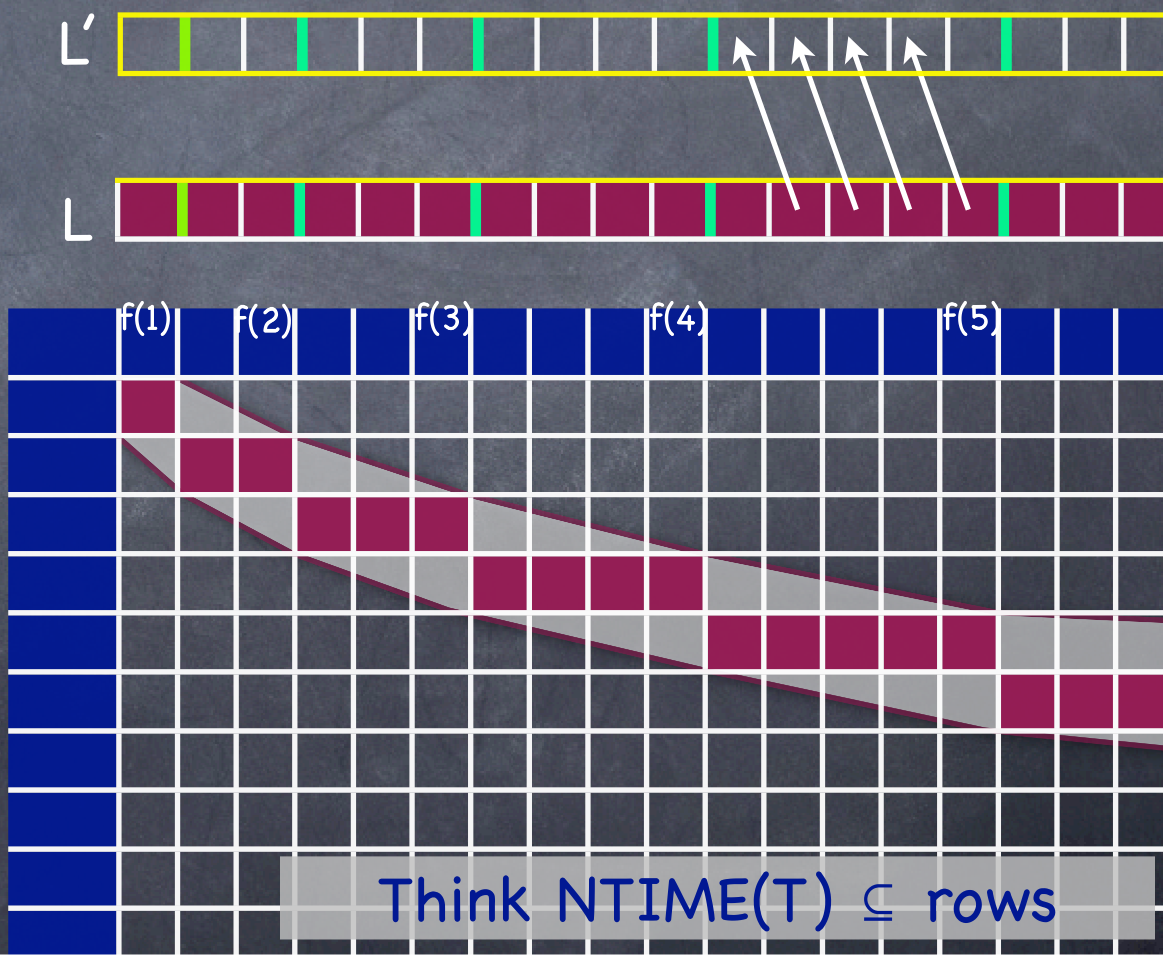
NTIME Hierarchy

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



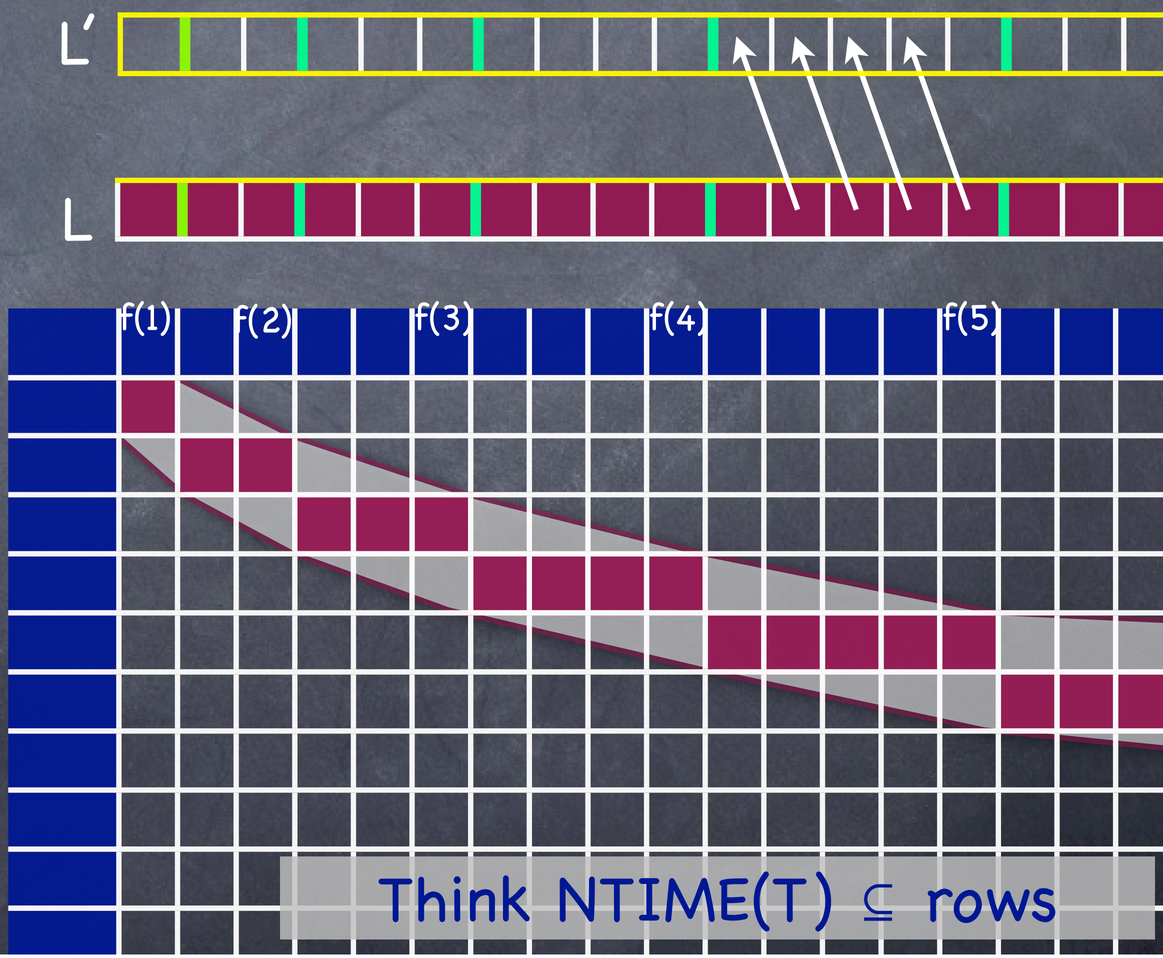
NTIME Hierarchy

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



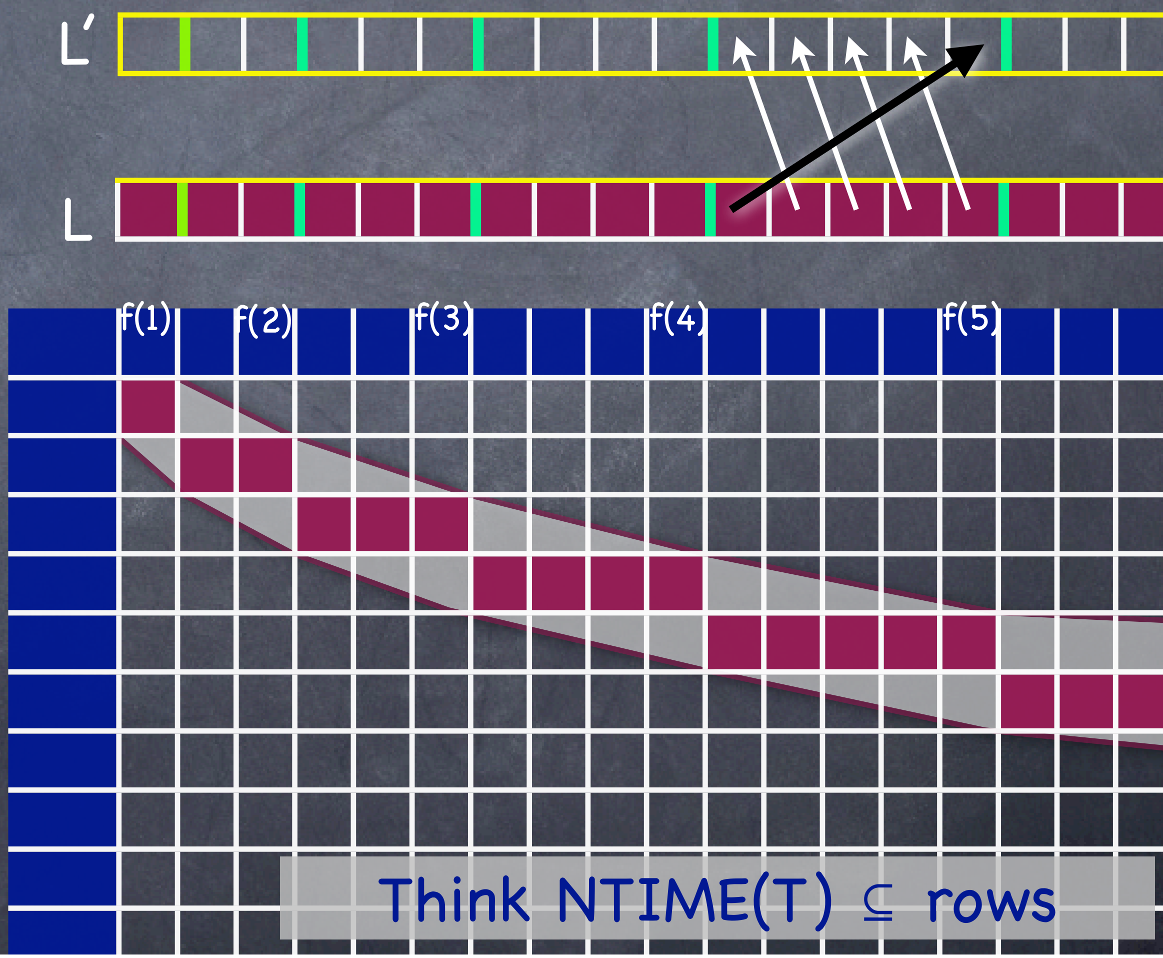
NTIME Hierarchy

- "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 $L'(j) = 1 - L(f(i-1)+1)$



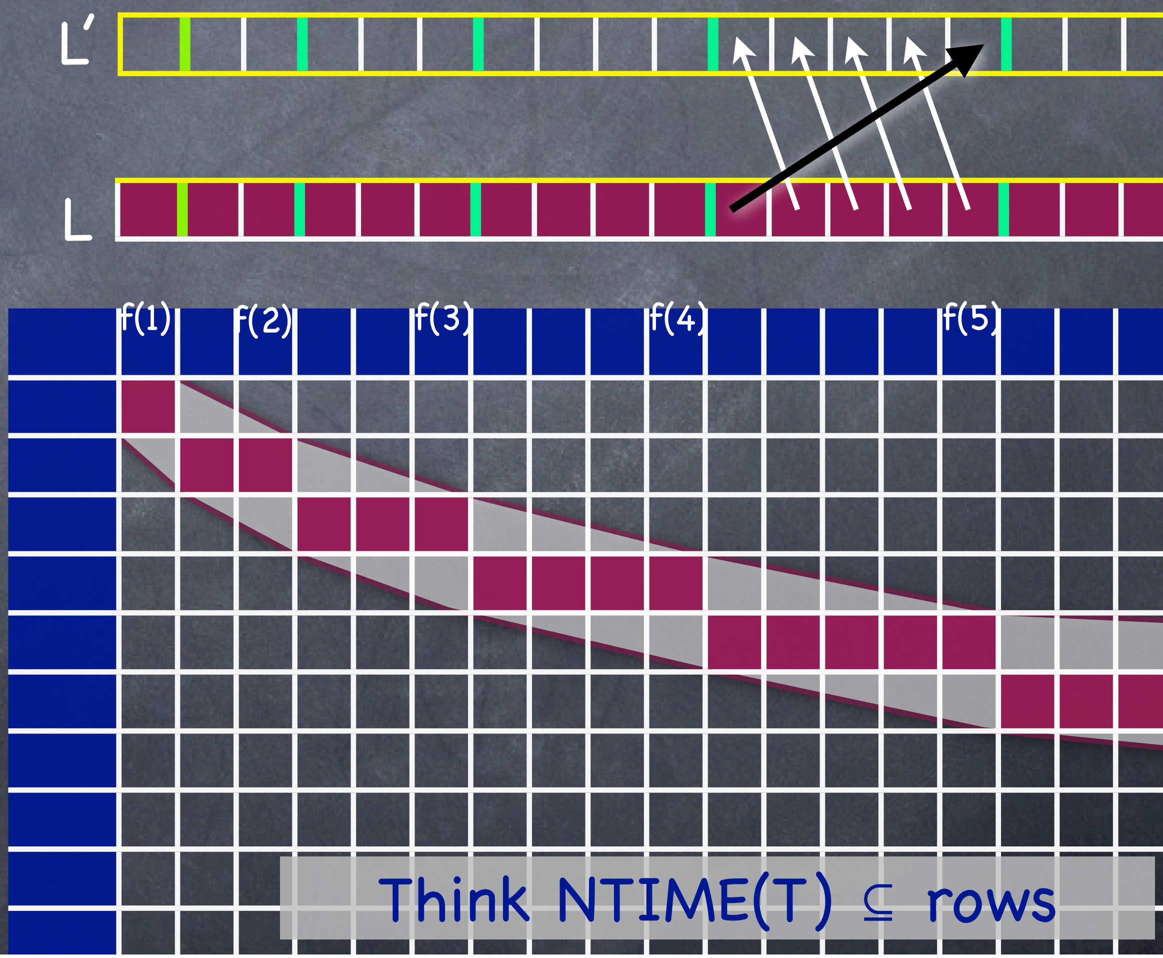
NTIME Hierarchy

- "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

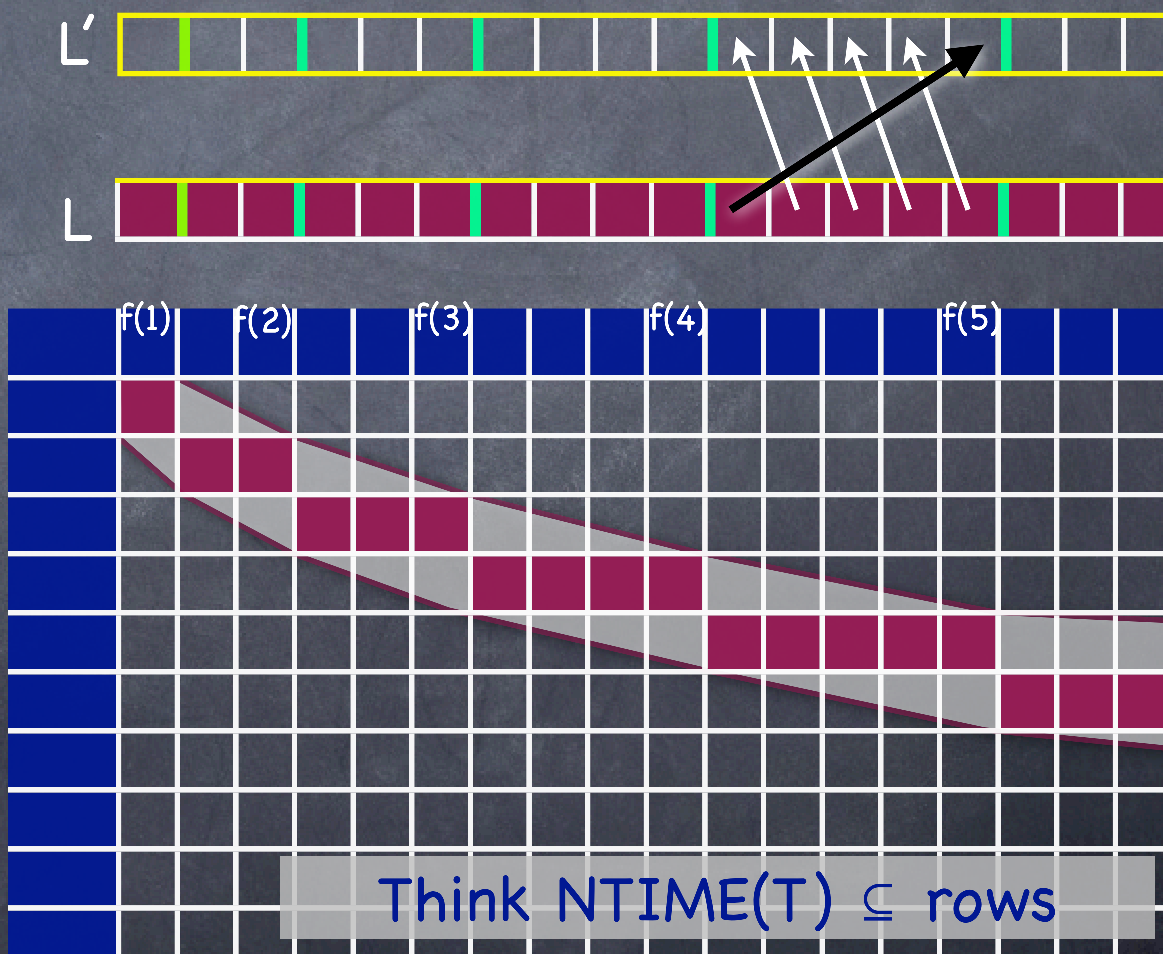
- "Delayed flip" on a "rapidly thickening diagonal"
  - $f(i+1)=\exp(f(i))$
  - Let L be the "diagonal" language
  - $L'(j)=L(j+1)$
    - except if $j=f(i)$, then $L'(j) = 1 - L(f(i-1)+1)$
  - L′ not in NTIME(T), but is in NTIME(T′)



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′)

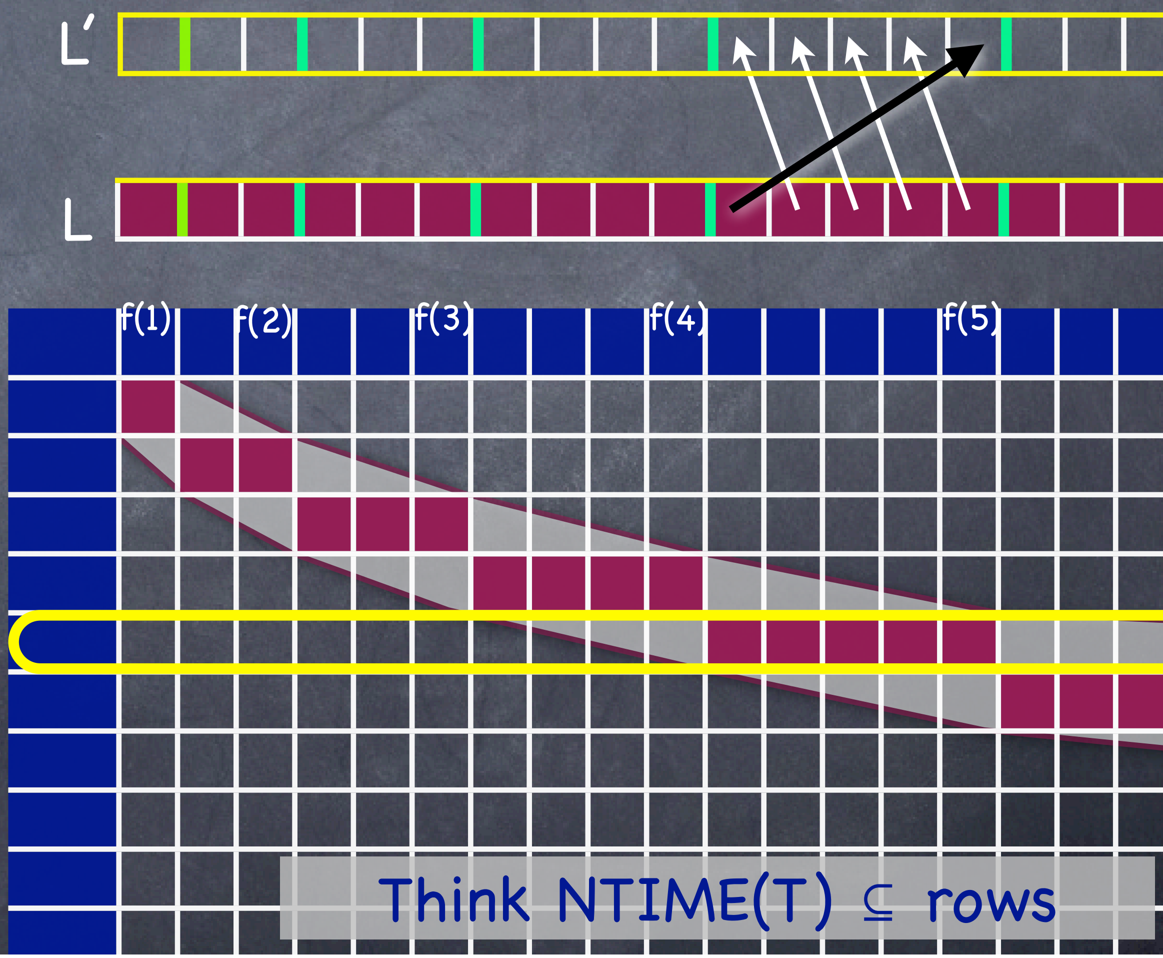
Flip, Diagonal



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 $L'(j) = 1 - L(f(i-1)+1)$
  - L′ not in NTIME(T), but is in NTIME(T′)

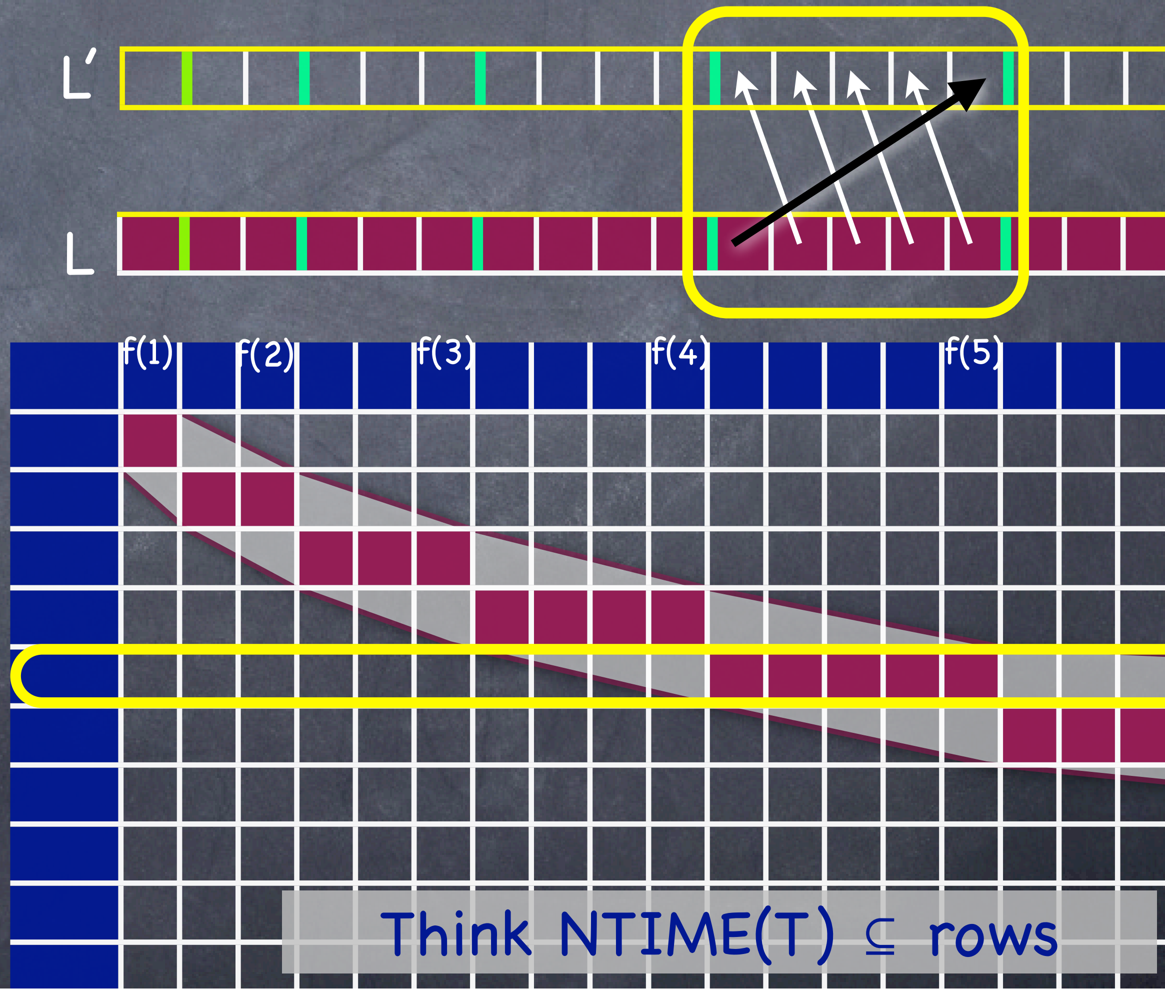
Flip, Diagonal



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

Flip, Diagonal

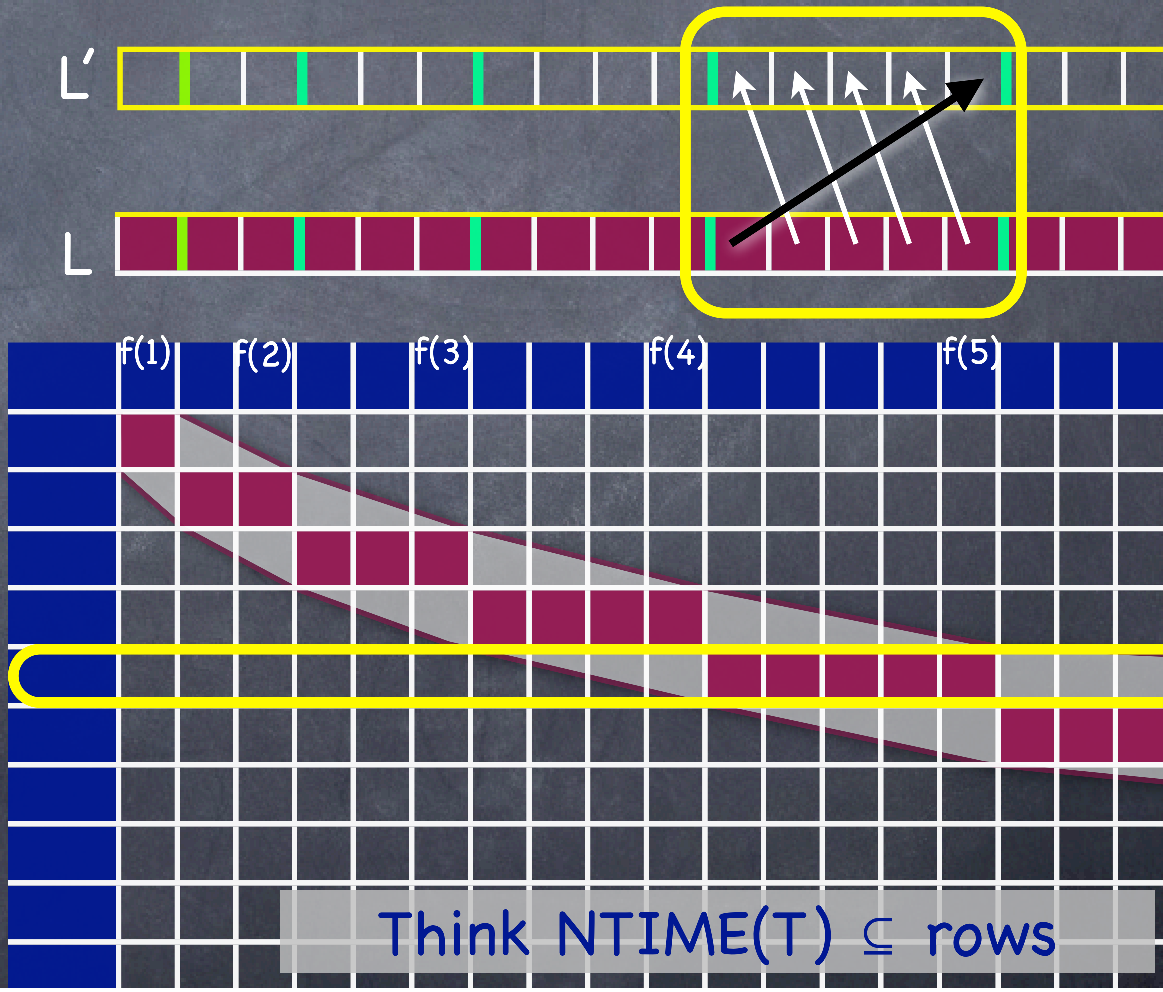


NTIME Hierarchy

- "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)
  - 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
  - DTIME(T) $\subsetneq$ DTIME(T′) if T log T = o(T′)
- NTIME Hierarchy
  - NTIME(T) $\subsetneq$ 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