

بسم الله الرحمن الرحيم

نظريه علوم کامپیوتر

نظريه علوم کامپیوتر - بهار ۱۴۰۰-۱۴۰۱ - جلسه سیزدهم: پیچیدگی حافظه (۲)

Theory of computation - 002 - S13 - space complexity (2)

Review: SPACE Complexity

Defn: Let $f: \mathbb{N} \rightarrow \mathbb{N}$ where $f(n) \geq n$. Say TM M runs in space $f(n)$ if M always halts and uses at most $f(n)$ tape cells on all inputs of length n .

Review: SPACE Complexity

Defn: Let $f: \mathbb{N} \rightarrow \mathbb{N}$ where $f(n) \geq n$. Say TM M runs in space $f(n)$ if M always halts and uses at most $f(n)$ tape cells on all inputs of length n .

An NTM M runs in space $f(n)$ if all branches halt and each branch uses at most $f(n)$ tape cells on all inputs of length n .

Review: SPACE Complexity

Defn: Let $f: \mathbb{N} \rightarrow \mathbb{N}$ where $f(n) \geq n$. Say TM M runs in space $f(n)$ if M always halts and uses at most $f(n)$ tape cells on all inputs of length n .

An NTM M runs in space $f(n)$ if all branches halt and each branch uses at most $f(n)$ tape cells on all inputs of length n .

$\text{SPACE}(f(n)) = \{B \mid \text{some 1-tape TM decides } B \text{ in space } O(f(n))\}$

$\text{NSPACE}(f(n)) = \{B \mid \text{some 1-tape NTM decides } B \text{ in space } O(f(n))\}$

Review: SPACE Complexity

Defn: Let $f: \mathbb{N} \rightarrow \mathbb{N}$ where $f(n) \geq n$. Say TM M runs in space $f(n)$ if M always halts and uses at most $f(n)$ tape cells on all inputs of length n .

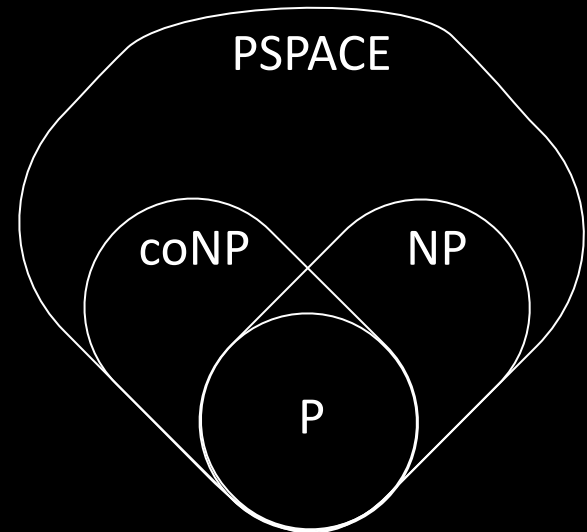
An NTM M runs in space $f(n)$ if all branches halt and each branch uses at most $f(n)$ tape cells on all inputs of length n .

$\text{SPACE}(f(n)) = \{B \mid \text{some 1-tape TM decides } B \text{ in space } O(f(n))\}$

$\text{NSPACE}(f(n)) = \{B \mid \text{some 1-tape NTM decides } B \text{ in space } O(f(n))\}$

$\text{PSPACE} = \bigcup_k \text{SPACE}(n^k)$ “polynomial space”

$\text{NPSPACE} = \bigcup_k \text{NSPACE}(n^k)$ “nondeterministic polynomial space”



Or possibly:

$$P = NP = \text{coNP} = \text{PSPACE}$$

Review: SPACE Complexity

Defn: Let $f: \mathbb{N} \rightarrow \mathbb{N}$ where $f(n) \geq n$. Say TM M runs in space $f(n)$ if M always halts and uses at most $f(n)$ tape cells on all inputs of length n .

An NTM M runs in space $f(n)$ if all branches halt and each branch uses at most $f(n)$ tape cells on all inputs of length n .

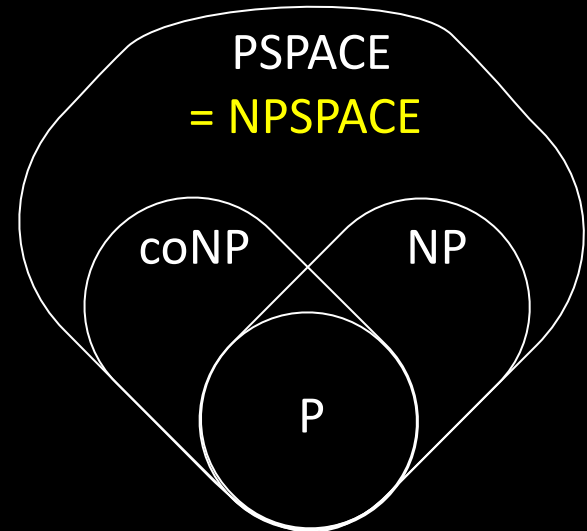
$\text{SPACE}(f(n)) = \{B \mid \text{some 1-tape TM decides } B \text{ in space } O(f(n))\}$

$\text{NSPACE}(f(n)) = \{B \mid \text{some 1-tape NTM decides } B \text{ in space } O(f(n))\}$

$\text{PSPACE} = \bigcup_k \text{SPACE}(n^k)$ “polynomial space”

$\text{NPSPACE} = \bigcup_k \text{NSPACE}(n^k)$ “nondeterministic polynomial space”

Today: $\text{PSPACE} = \text{NPSPACE}$



Or possibly:

$P = NP = \text{coNP} = \text{PSPACE}$

Review: SPACE Complexity

Defn: Let $f: \mathbb{N} \rightarrow \mathbb{N}$ where $f(n) \geq n$. Say TM M runs in space $f(n)$ if M always halts and uses at most $f(n)$ tape cells on all inputs of length n .

An NTM M runs in space $f(n)$ if all branches halt and each branch uses at most $f(n)$ tape cells on all inputs of length n .

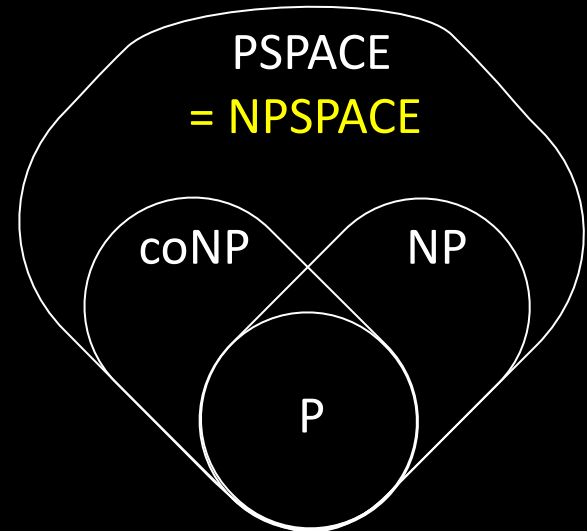
$\text{SPACE}(f(n)) = \{B \mid \text{some 1-tape TM decides } B \text{ in space } O(f(n))\}$

$\text{NSPACE}(f(n)) = \{B \mid \text{some 1-tape NTM decides } B \text{ in space } O(f(n))\}$

$\text{PSPACE} = \bigcup_k \text{SPACE}(n^k)$ “polynomial space”

$\text{NPSPACE} = \bigcup_k \text{NSPACE}(n^k)$ “nondeterministic polynomial space”

Today: $\text{PSPACE} = \text{NPSPACE}$



Or possibly:

$P = NP = \text{coNP} = \text{PSPACE}$

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A .

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A .

WORK
PORK
PORT

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A .

WORK
PORK
PORT
SORT

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A .

WORK
PORK
PORT
SORT
SOOT

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A .

WORK
PORK
PORT
SORT
SOOT
SLOT

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A .

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A .

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A .

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A .

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A .

Defn: $LADDERDFA = \{ \langle B, u, v \rangle \mid B \text{ is a DFA and } L(B) \text{ contains a ladder } y_1, y_2, \dots, y_k \text{ where } y_1 = u \text{ and } y_k = v \}.$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A .

Defn: $LADDERDFA = \{ \langle B, u, v \rangle \mid B \text{ is a DFA and } L(B) \text{ contains a ladder } y_1, y_2, \dots, y_k \text{ where } y_1 = u \text{ and } y_k = v \}.$

Theorem: $LADDERDFA \in \text{NPSPACE}$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

Example: Ladder Problem

A ladder is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A word ladder for English is a ladder of English words.

Let A be a language. A ladder in A is a ladder of strings in A .

Defn: $LADDERDFA = \{ \langle B, u, v \rangle \mid B \text{ is a DFA and } L(B) \text{ contains a ladder } y_1, y_2, \dots, y_k \text{ where } y_1 = u \text{ and } y_k = v \}$.

Theorem: $LADDERDFA \in \text{NPSPACE}$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

*LADDER*DFA \in NPSPACE

Theorem: *LADDER*DFA \in NPSPACE

*LADDER*DFA \in NPSPACE

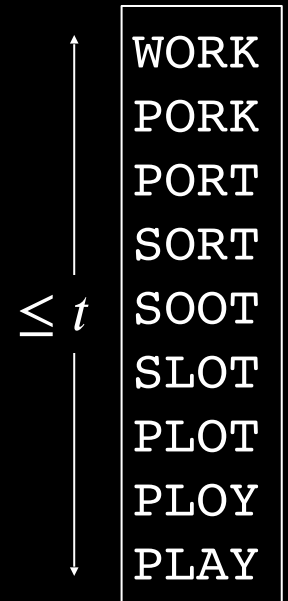
Theorem: *LADDER*DFA \in NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v .

*LADDER*DFA \in NPSPACE

Theorem: *LADDER*DFA \in NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v .

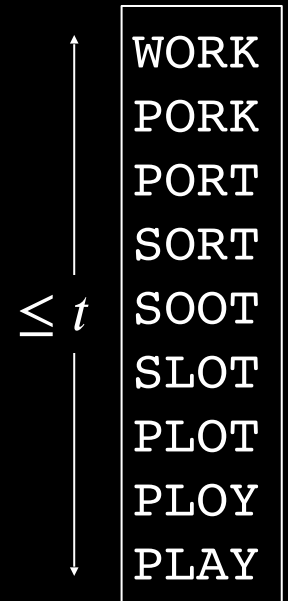


*LADDER*DFA \in NPSPACE

Theorem: *LADDER*DFA \in NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.



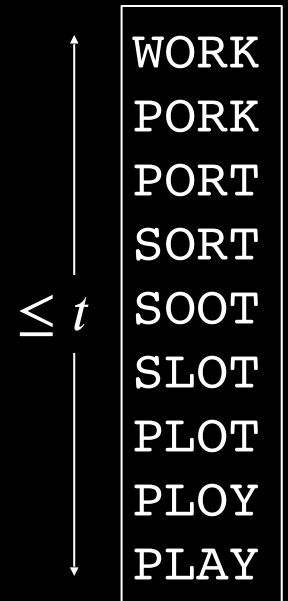
*LADDER*DFA \in NPSPACE

Theorem: *LADDER*DFA \in NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

Proof: “On input $\langle B, u, v \rangle$



*LADDER*DFA \in NPSPACE

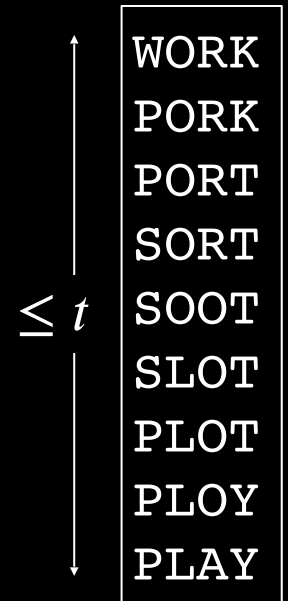
Theorem: *LADDER*DFA \in NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.



*LADDER*DFA \in NPSPACE

Theorem: *LADDER*DFA \in NPSPACE

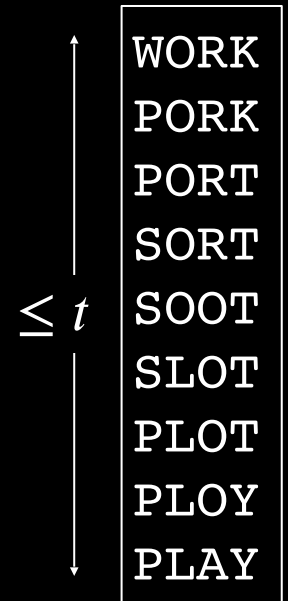
Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.

2. Repeat at most t times where $t = |\Sigma|^m$.



*LADDER*DFA \in NPSPACE

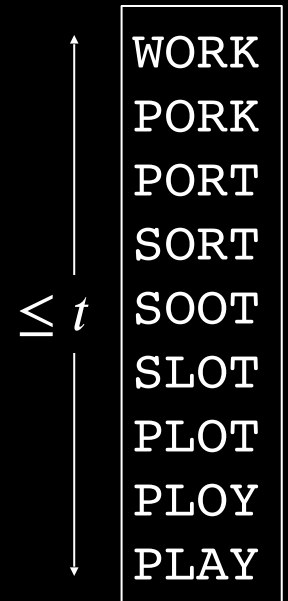
Theorem: *LADDER*DFA \in NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.
2. Repeat at most t times where $t = |\Sigma|^m$.
3. Nondeterministically change one symbol in y .



*LADDER*DFA \in NPSPACE

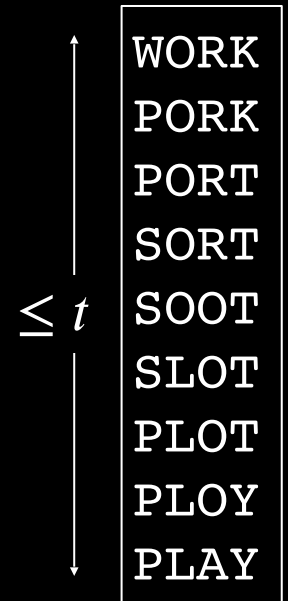
Theorem: *LADDER*DFA \in NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.
2. Repeat at most t times where $t = |\Sigma|^m$.
3. Nondeterministically change one symbol in y .
4. *Reject* if $y \notin L(B)$.



*LADDER*DFA \in NPSPACE

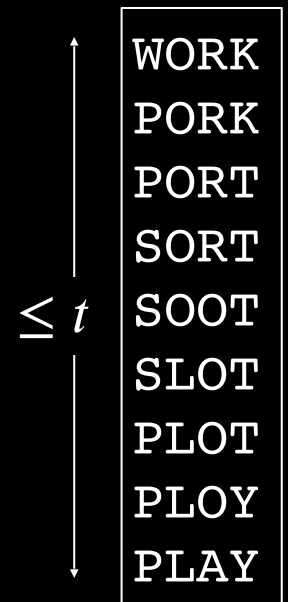
Theorem: *LADDER*DFA \in NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.
2. Repeat at most t times where $t = |\Sigma|^m$.
3. Nondeterministically change one symbol in y .
4. *Reject* if $y \notin L(B)$.
5. *Accept* if $y = v$.



*LADDER*DFA \in NPSPACE

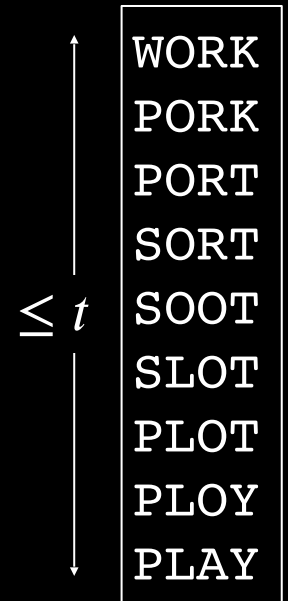
Theorem: *LADDER*DFA \in NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.
2. Repeat at most t times where $t = |\Sigma|^m$.
3. Nondeterministically change one symbol in y .
4. *Reject* if $y \notin L(B)$.
5. *Accept* if $y = v$.
6. *Reject* [exceeded t steps].



LADDERDFA \in NPSPACE

Theorem: *LADDERDFA* \in NPSPACE

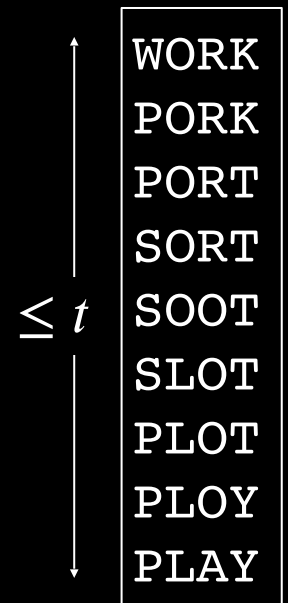
Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.
2. Repeat at most t times where $t = |\Sigma|^m$.
3. Nondeterministically change one symbol in y .
4. *Reject* if $y \notin L(B)$.
5. *Accept* if $y = v$.
6. *Reject* [exceeded t steps].

Space used is for storing y and t .



LADDERDFA \in NPSPACE

Theorem: *LADDERDFA* \in NPSPACE

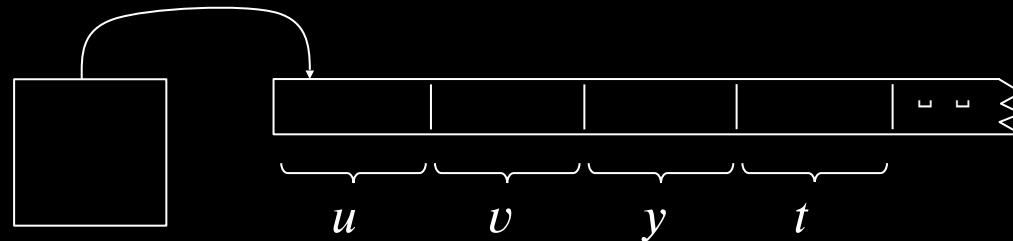
Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.
2. Repeat at most t times where $t = |\Sigma|^m$.
3. Nondeterministically change one symbol in y .
4. *Reject* if $y \notin L(B)$.
5. *Accept* if $y = v$.
6. *Reject* [exceeded t steps].

Space used is for storing y and t .



$\leq t$	WORK
	PORK
	PORT
	SORT
	SOOT
	SLOT
	PLOT
	PLAY

LADDERDFA \in NPSPACE

Theorem: *LADDERDFA* \in NPSPACE

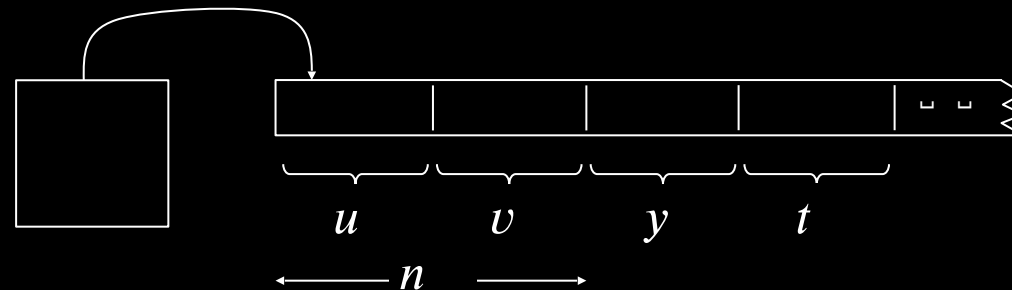
Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.
2. Repeat at most t times where $t = |\Sigma|^m$.
3. Nondeterministically change one symbol in y .
4. *Reject* if $y \notin L(B)$.
5. *Accept* if $y = v$.
6. *Reject* [exceeded t steps].

Space used is for storing y and t .



LADDERDFA \in NPSPACE

Theorem: *LADDERDFA* \in NPSPACE

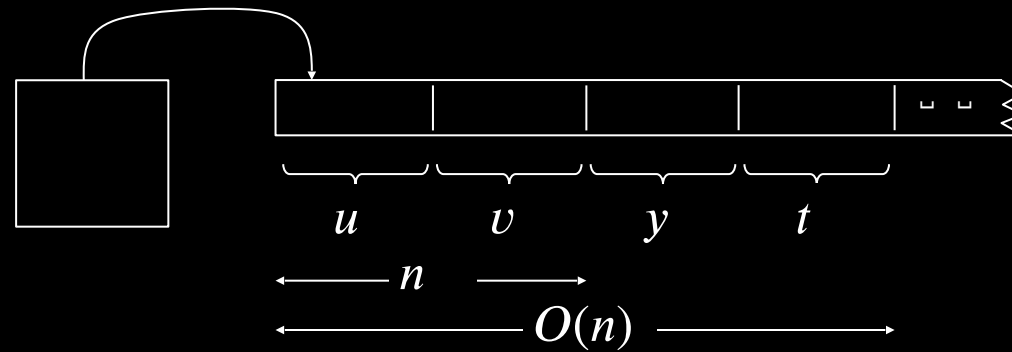
Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.
2. Repeat at most t times where $t = |\Sigma|^m$.
3. Nondeterministically change one symbol in y .
4. *Reject* if $y \notin L(B)$.
5. *Accept* if $y = v$.
6. *Reject* [exceeded t steps].

Space used is for storing y and t .



LADDERDFA \in NPSPACE

Theorem: *LADDERDFA* \in NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v .

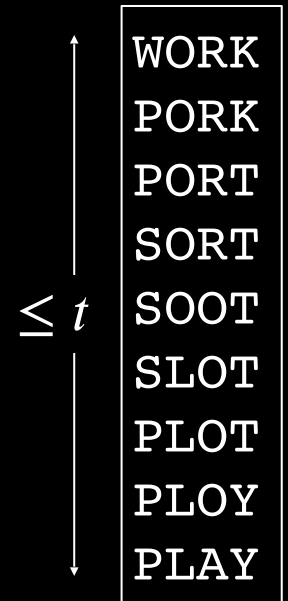
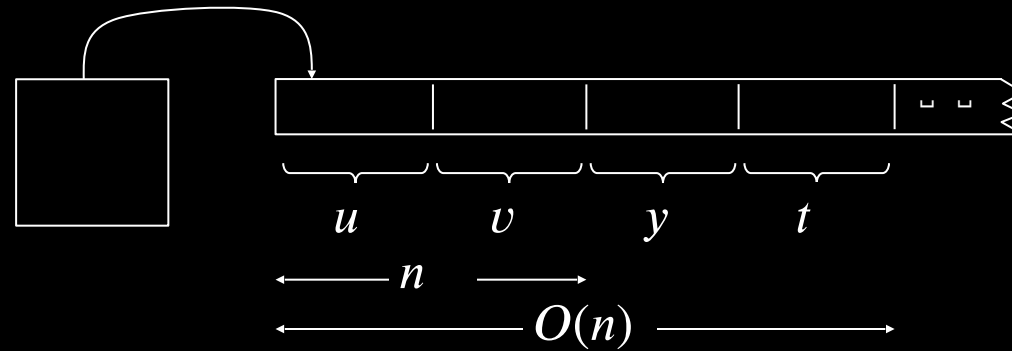
Careful- (a) cannot store sequence, (b) must terminate.

Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.
2. Repeat at most t times where $t = |\Sigma|^m$.
3. Nondeterministically change one symbol in y .
4. *Reject* if $y \notin L(B)$.
5. *Accept* if $y = v$.
6. *Reject* [exceeded t steps].

Space used is for storing y and t .

LADDERDFA \in NPSPACE(n).



LADDERDFA \in NPSPACE

Theorem: *LADDERDFA* \in NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

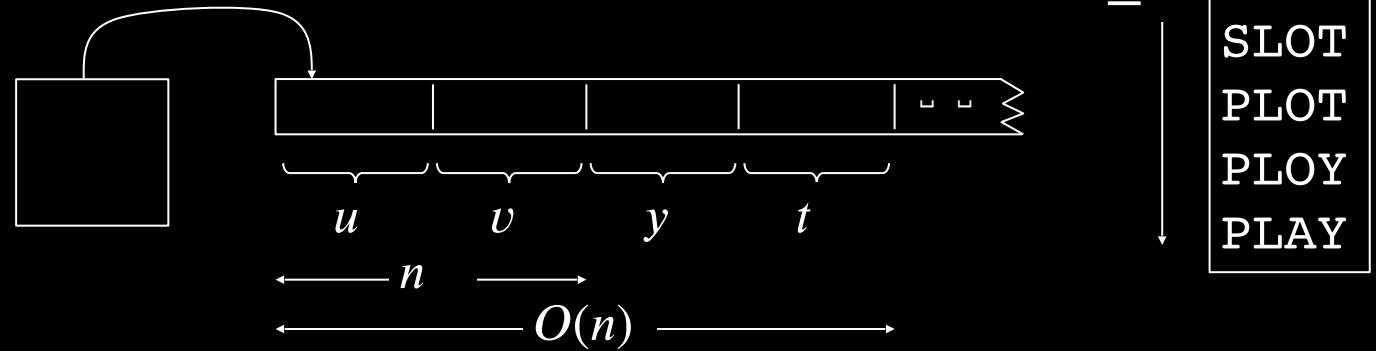
Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.
2. Repeat at most t times where $t = |\Sigma|^m$.
3. Nondeterministically change one symbol in y .
4. *Reject* if $y \notin L(B)$.
5. *Accept* if $y = v$.
6. *Reject* [exceeded t steps].

Space used is for storing y and t .

LADDERDFA \in NSPACE(n).

Theorem: *LADDERDFA* \in PSPACE (!)



LADDERDFA \in NPSPACE

Theorem: *LADDERDFA* \in NPSPACE

Proof idea: Nondeterministically guess the sequence from u to v .

Careful- (a) cannot store sequence, (b) must terminate.

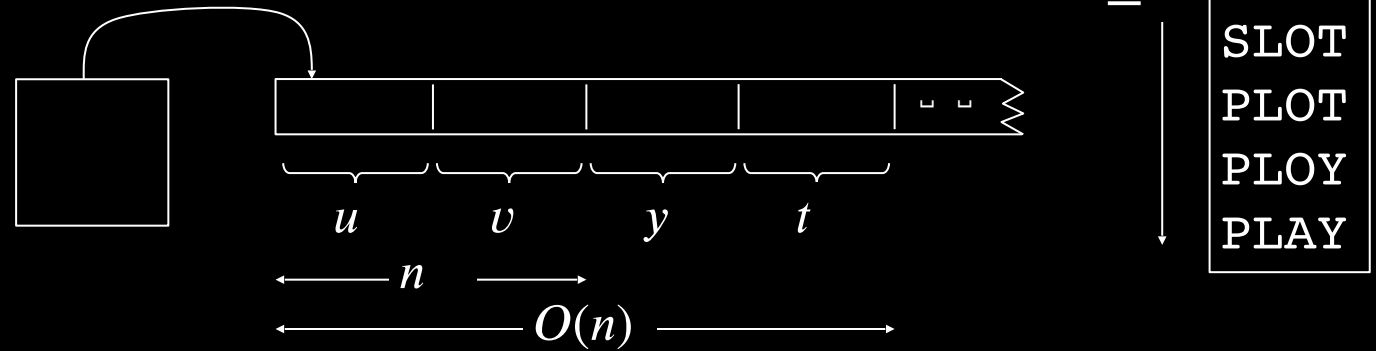
Proof: “On input $\langle B, u, v \rangle$

1. Let $y = u$ and let $m = |u|$.
2. Repeat at most t times where $t = |\Sigma|^m$.
3. Nondeterministically change one symbol in y .
4. *Reject* if $y \notin L(B)$.
5. *Accept* if $y = v$.
6. *Reject* [exceeded t steps].

Space used is for storing y and t .

LADDERDFA \in NSPACE(n).

Theorem: *LADDERDFA* \in PSPACE (!)



LADDERDFA \in PSPACE

$LADDERDFA \in PSPACE$

Theorem: $LADDERDFA \in SPACE(n^2)$

LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

$LADDERDFA \in PSPACE$

Theorem: $LADDERDFA \in SPACE(n^2)$

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

*LADDER*DFA \in PSPACE

Theorem: *LADDER*DFA \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

*BOUNDED-LADDER*DFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

WORK

PLAY

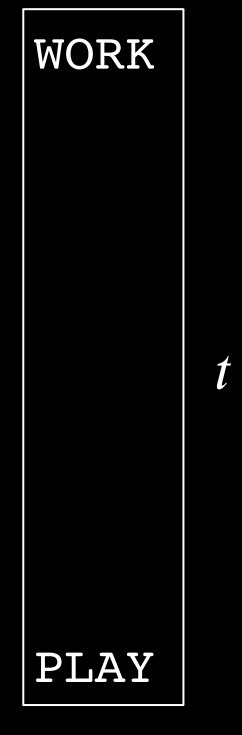
LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$



*LADDER*DFA \in PSPACE

Theorem: *LADDER*DFA \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

*BOUNDED-LADDER*DFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$



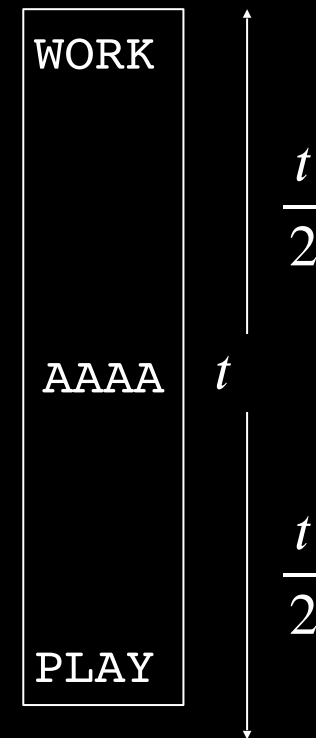
*LADDER*DFA \in PSPACE

Theorem: *LADDER*DFA \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

*BOUNDED-LADDER*DFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$



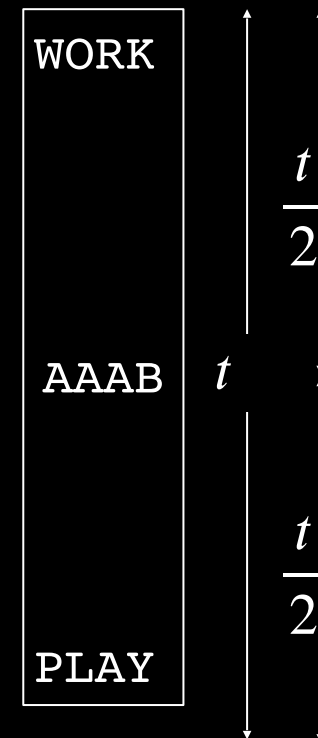
*LADDER*DFA \in PSPACE

Theorem: *LADDER*DFA \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

*BOUNDED-LADDER*DFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$



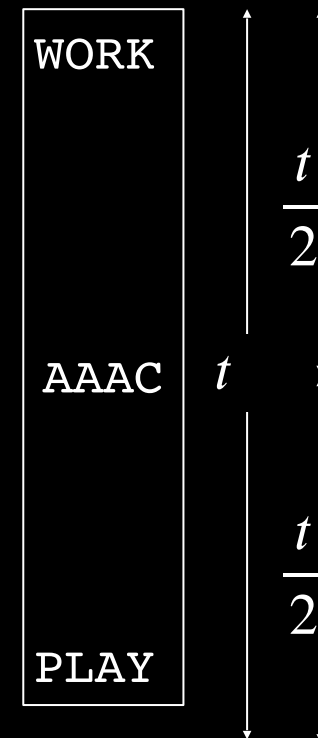
*LADDER*DFA \in PSPACE

Theorem: *LADDER*DFA \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

*BOUNDED-LADDER*DFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$



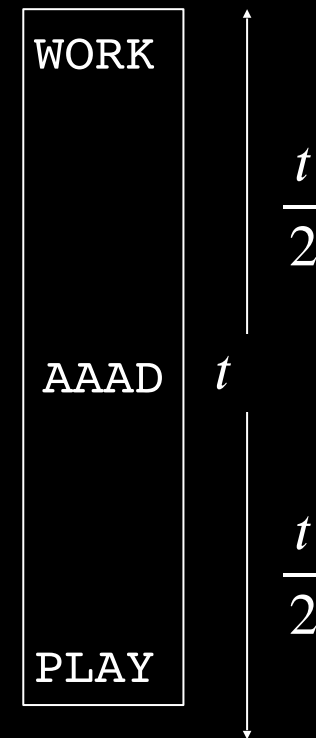
*LADDER*DFA \in PSPACE

Theorem: *LADDER*DFA \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

*BOUNDED-LADDER*DFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$



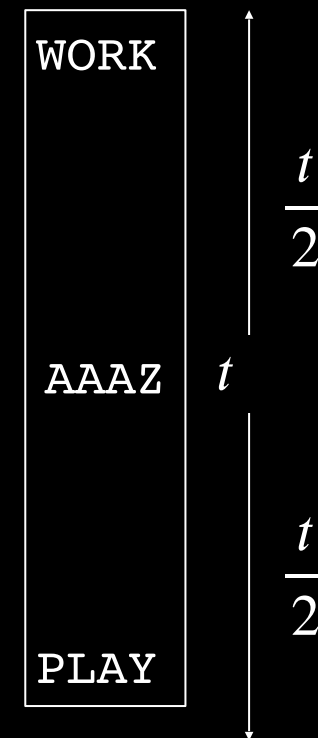
*LADDER*DFA \in PSPACE

Theorem: *LADDER*DFA \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

*BOUNDED-LADDER*DFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$



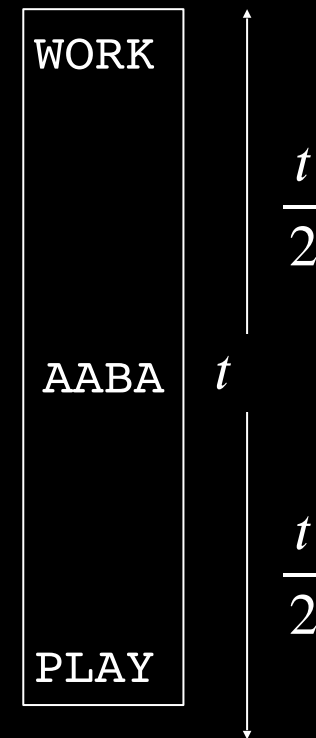
*LADDER*DFA \in PSPACE

Theorem: *LADDER*DFA \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

*BOUNDED-LADDER*DFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$



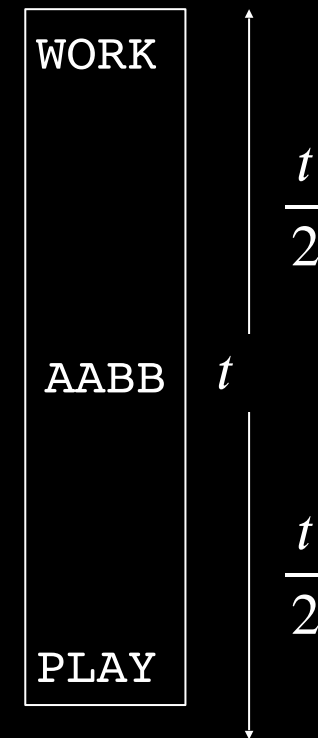
*LADDER*DFA \in PSPACE

Theorem: *LADDER*DFA \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

*BOUNDED-LADDER*DFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$



*LADDER*DFA \in PSPACE

Theorem: *LADDER*DFA \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

*BOUNDED-LADDER*DFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$



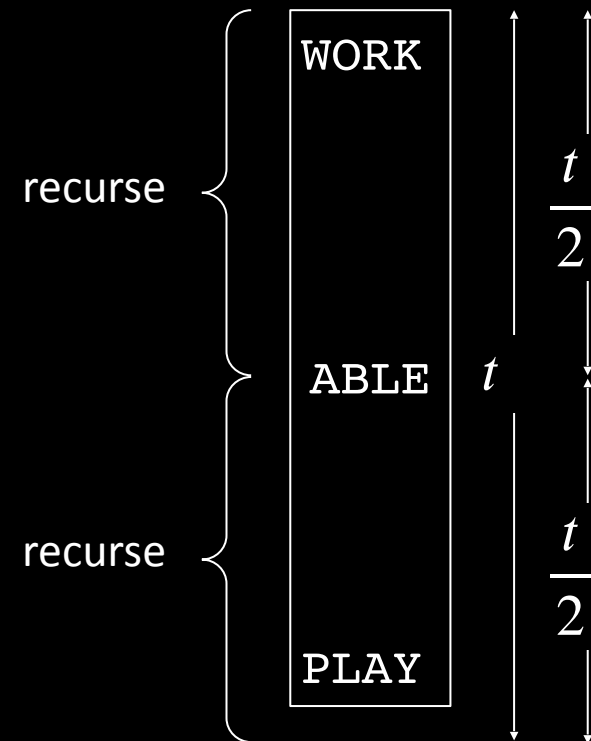
*LADDER*DFA \in PSPACE

Theorem: *LADDER*DFA \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

*BOUNDED-LADDER*DFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$



LADDERDFA \in PSPACE

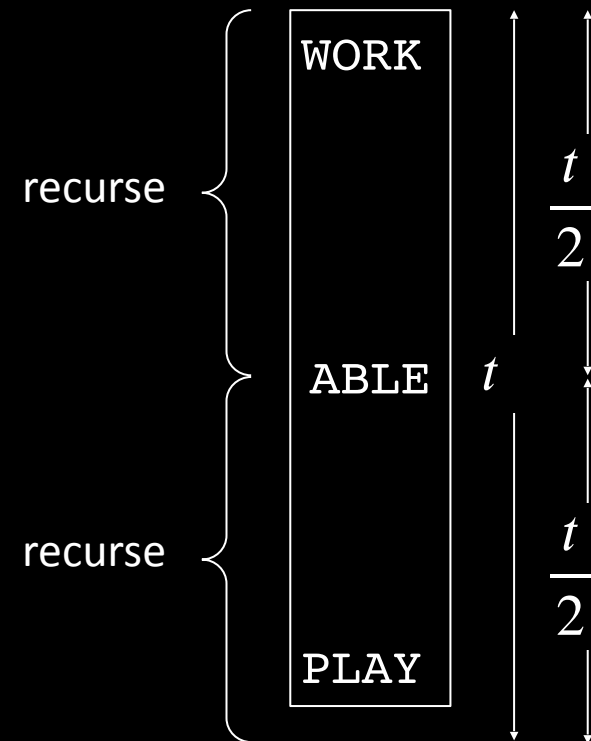
Theorem: *LADDERDFA* \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

B-L = "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.



LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

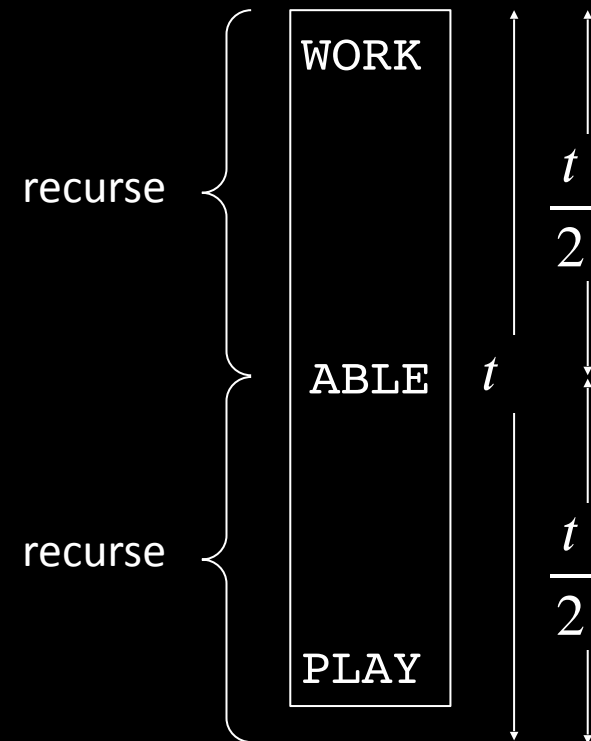
Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

B-L = "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.

1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.



LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

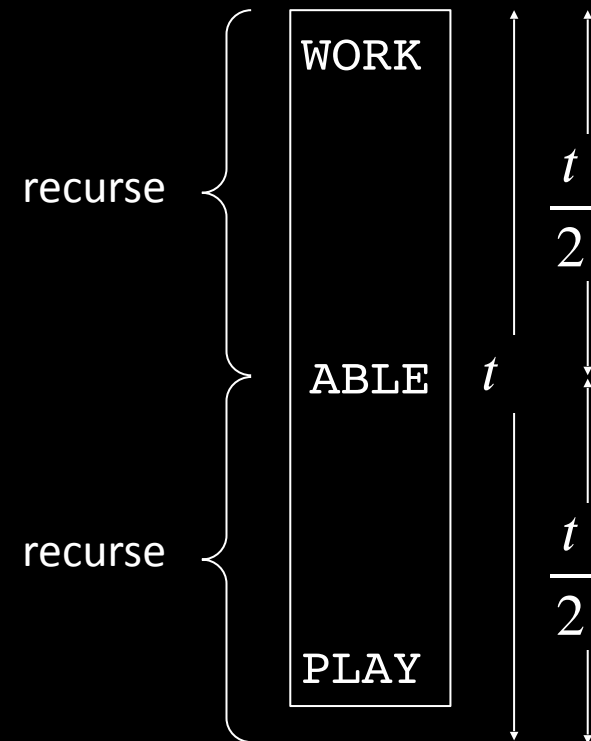
Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

B-L = "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.

1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.
2. For $b > 1$, repeat for each w of length $|u|$



LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

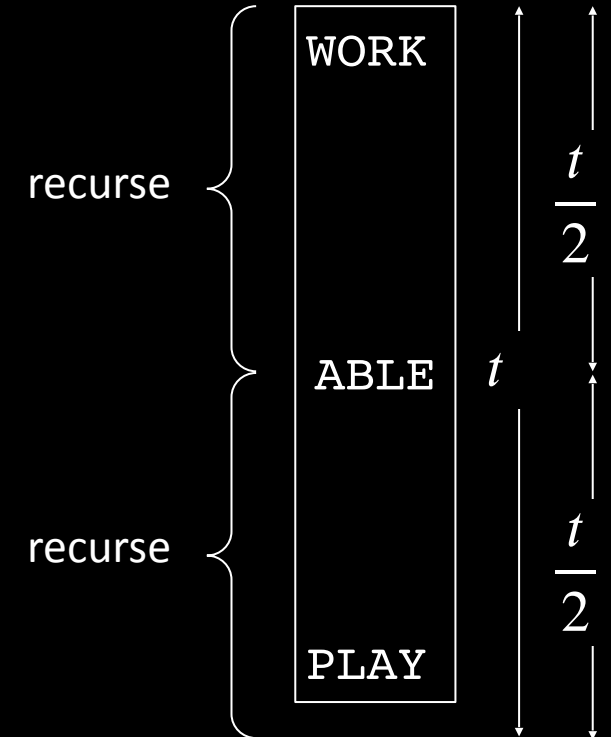
Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

B-L = "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.

1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.
2. For $b > 1$, repeat for each w of length $|u|$
3. Recursively test $u \xrightarrow{b/2} w$ and $w \xrightarrow{b/2} v$ [division rounds up]



LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

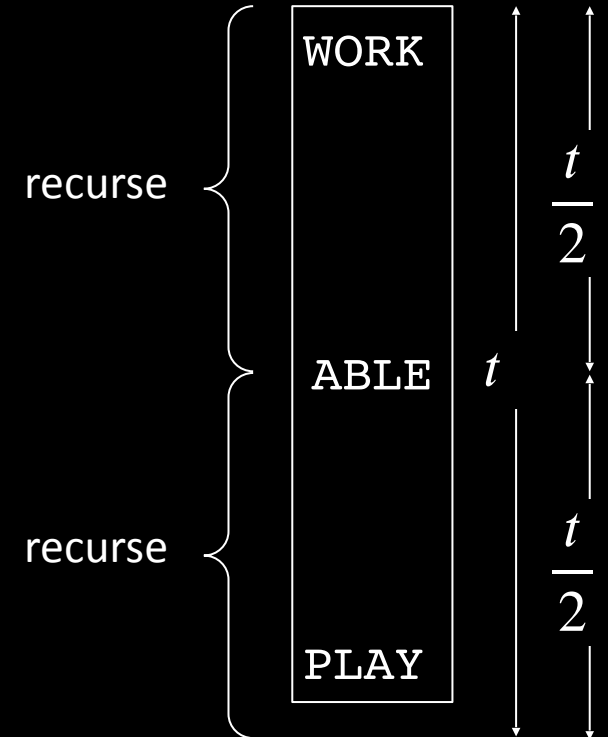
Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

B-L = "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.

1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.
2. For $b > 1$, repeat for each w of length $|u|$
3. Recursively test $u \xrightarrow{b/2} w$ and $w \xrightarrow{b/2} v$ [division rounds up]
4. *Accept* both accept.



LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

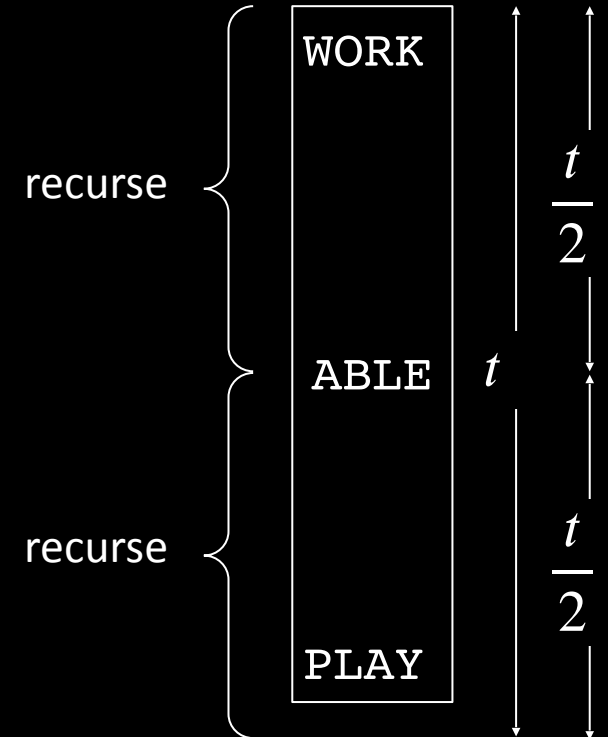
Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

B-L = "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.

1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.
2. For $b > 1$, repeat for each w of length $|u|$
3. Recursively test $u \xrightarrow{b/2} w$ and $w \xrightarrow{b/2} v$ [division rounds up]
4. *Accept* both accept.
5. *Reject* [if all fail]."



LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

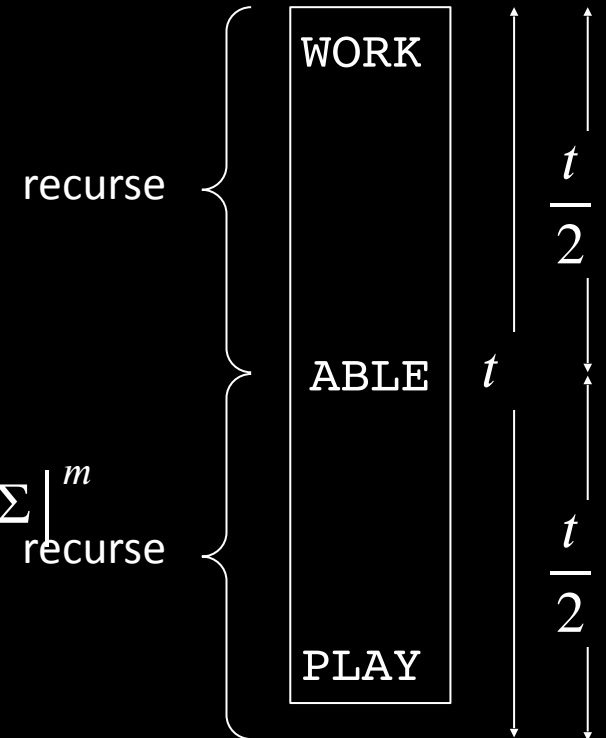
Here's a recursive procedure to solve the bounded DFA ladder problem:

BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

B-L = "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.

1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.
2. For $b > 1$, repeat for each w of length $|u|$
 3. Recursively test $u \xrightarrow{b/2} w$ and $w \xrightarrow{b/2} v$ [division rounds up]
 4. *Accept* both accept.
 5. *Reject* [if all fail]."

Test $\langle B, u, v \rangle \in \textit{LADDERDFA}$ with *B-L* procedure on input $\langle B, u, v, t \rangle$ for $t = \left| \Sigma \right|_{\text{recurse}}^m$



LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

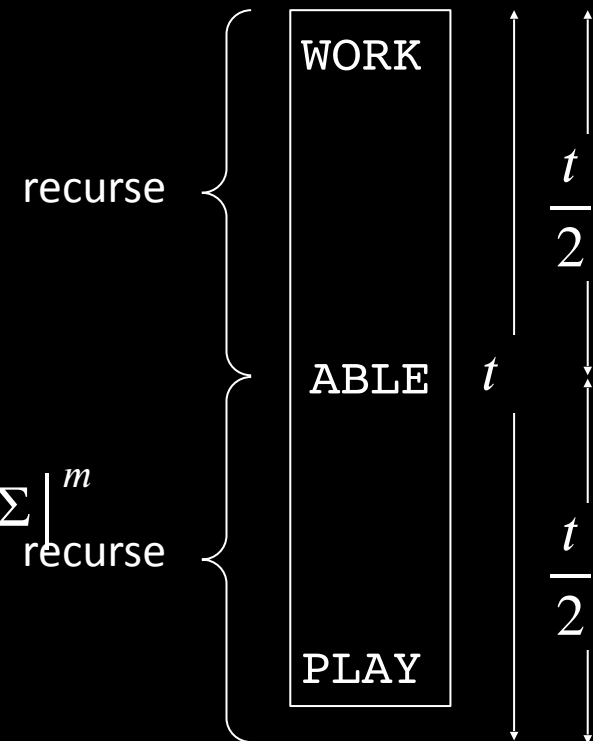
BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

B-L = "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.

1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.
2. For $b > 1$, repeat for each w of length $|u|$
 3. Recursively test $u \xrightarrow{b/2} w$ and $w \xrightarrow{b/2} v$ [division rounds up]
 4. *Accept* both accept.
 5. *Reject* [if all fail]."

Test $\langle B, u, v \rangle \in \textit{LADDERDFA}$ with *B-L* procedure on input $\langle B, u, v, t \rangle$ for $t = \left| \Sigma \right|_{\text{recurse}}^m$

Space analysis:



LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

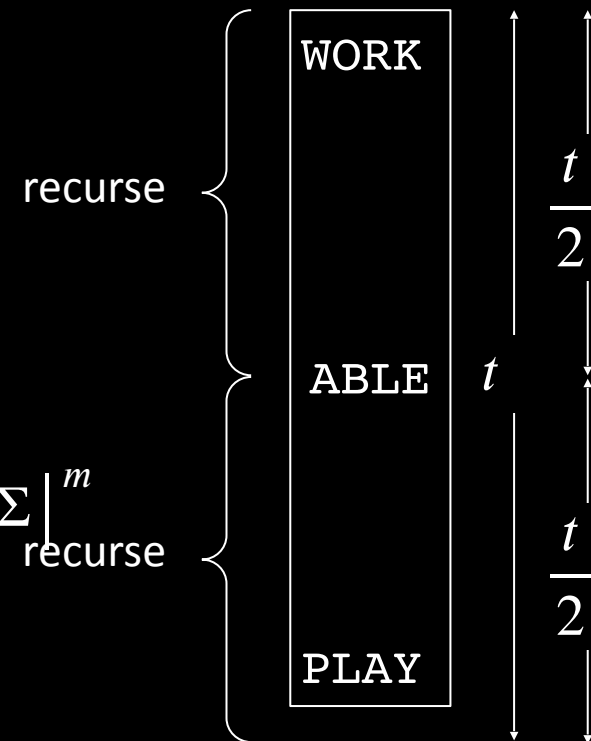
B-L = "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.

1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.
2. For $b > 1$, repeat for each w of length $|u|$
 3. Recursively test $u \xrightarrow{b/2} w$ and $w \xrightarrow{b/2} v$ [division rounds up]
 4. *Accept* both accept.
 5. *Reject* [if all fail]."

Test $\langle B, u, v \rangle \in \textit{LADDERDFA}$ with *B-L* procedure on input $\langle B, u, v, t \rangle$ for $t = \left| \Sigma \right|_{\text{recurse}}^m$

Space analysis:

Each recursive level uses space $O(n)$ (to record w).



$LADDERDFA \in PSPACE$

Theorem: $LADDERDFA \in SPACE(n^2)$

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

$BOUNDED-LADDERDFA = \{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

$B-L =$ "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.

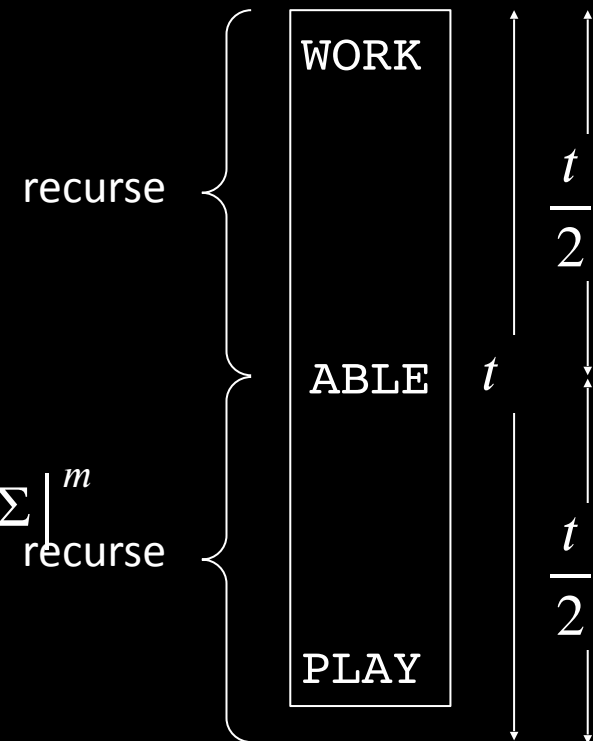
1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.
2. For $b > 1$, repeat for each w of length $|u|$
 3. Recursively test $u \xrightarrow{b/2} w$ and $w \xrightarrow{b/2} v$ [division rounds up]
 4. *Accept* both accept.
 5. *Reject* [if all fail]."

Test $\langle B, u, v \rangle \in LADDERDFA$ with $B-L$ procedure on input $\langle B, u, v, t \rangle$ for $t = \left| \Sigma \right|_m^{\text{recurse}}$

Space analysis:

Each recursive level uses space $O(n)$ (to record w).

Recursion depth is $\log t = O(m) = O(n)$.



LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

B-L = "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.

1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.
2. For $b > 1$, repeat for each w of length $|u|$
 3. Recursively test $u \xrightarrow{b/2} w$ and $w \xrightarrow{b/2} v$ [division rounds up]
 4. *Accept* both accept.
 5. *Reject* [if all fail]."

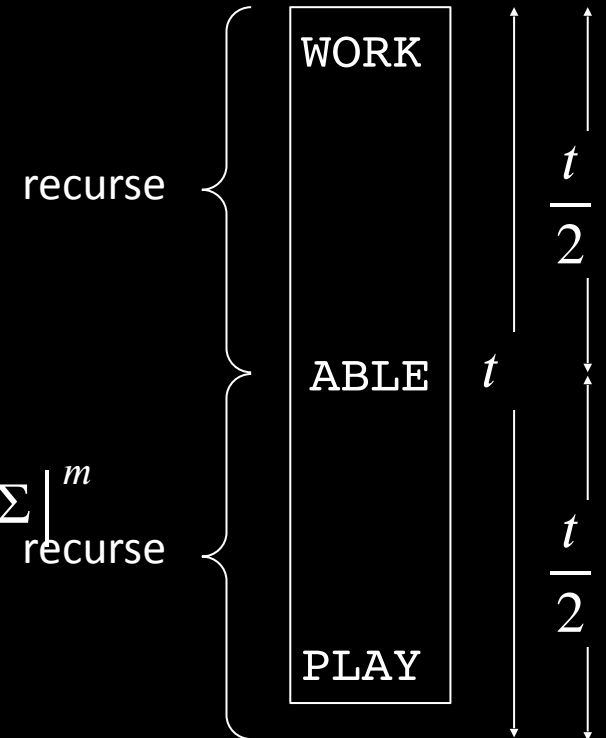
Test $\langle B, u, v \rangle \in \textit{LADDERDFA}$ with *B-L* procedure on input $\langle B, u, v, t \rangle$ for $t = \left| \Sigma \right|_m^{\text{recurse}}$

Space analysis:

Each recursive level uses space $O(n)$ (to record w).

Recursion depth is $\log t = O(m) = O(n)$.

Total space used is $O(n^2)$.



LADDERDFA \in PSPACE

Theorem: $LADDERDFA \in \text{SPACE}(n^2)$

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

$BOUNDED\text{-}LADDERDFA = \{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

$B\text{-}L =$ "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.

1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.
2. For $b > 1$, repeat for each w of length $|u|$
 3. Recursively test $u \xrightarrow{b/2} w$ and $w \xrightarrow{b/2} v$ [division rounds up]
 4. *Accept* both accept.
 5. *Reject* [if all fail]."

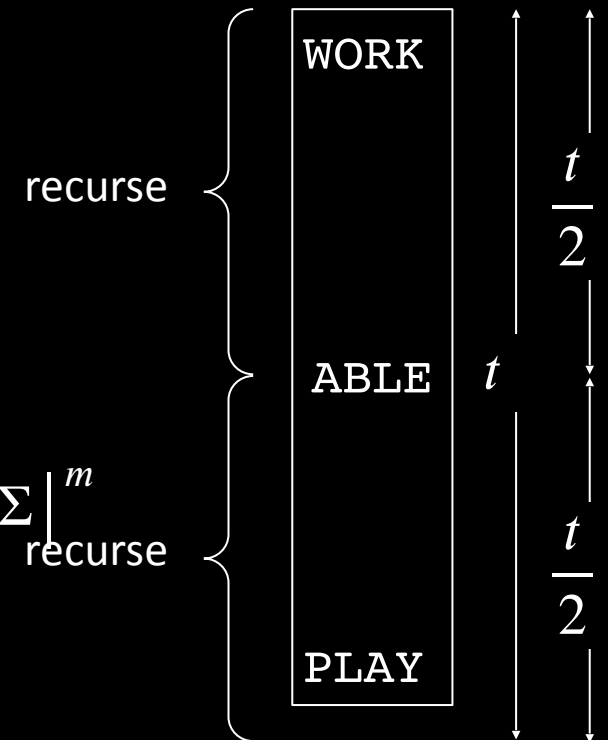
Test $\langle B, u, v \rangle \in LADDERDFA$ with $B\text{-}L$ procedure on input $\langle B, u, v, t \rangle$ for $t = \left| \sum_{\text{recurse}}^m \right|$

Space analysis:

Each recursive level uses space $O(n)$ (to record w).

Recursion depth is $\log t = O(m) = O(n)$.

Total space used is $O(n^2)$.



LADDERDFA \in PSPACE

Theorem: *LADDERDFA* \in SPACE(n^2)

Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.

Here's a recursive procedure to solve the bounded DFA ladder problem:

BOUNDED-LADDERDFA = $\{ \langle B, u, v, b \rangle \mid B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \}$

B-L = "On input $\langle B, u, v, b \rangle$ Let $m = |u| = |v|$.

1. For $b = 1$, *accept* if $u, v \in L(B)$ and differ in ≤ 1 place, else *reject*.
2. For $b > 1$, repeat for each w of length $|u|$
3. Recursively test $u \xrightarrow{b/2} w$ and $w \xrightarrow{b/2} v$ [division rounds up]
4. *Accept* both accept.
5. *Reject* [if all fail]."

Test $\langle B, u, v \rangle \in \textit{LADDERDFA}$ with *B-L* procedure on input $\langle B, u, v, t \rangle$ for t

Space analysis:

Each recursive level uses space $O(n)$ (to record w).

Recursion depth is $\log t = O(m) = O(n)$.

Total space used is $O(n^2)$.

Check-in 17.3

Find an English word ladder connecting MUST and VOTE.

(a) Already did it.

(b) I will.

PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

Proof: Convert NTM N to equivalent TM M , only squaring the space used.

PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

PSPACE = NPSPACE

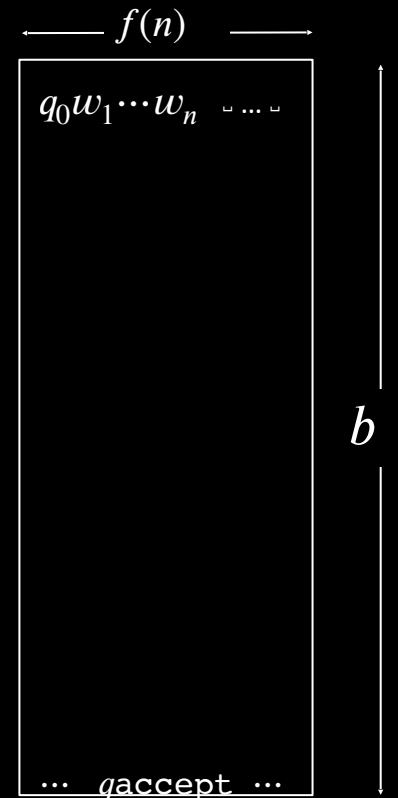
Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]



PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

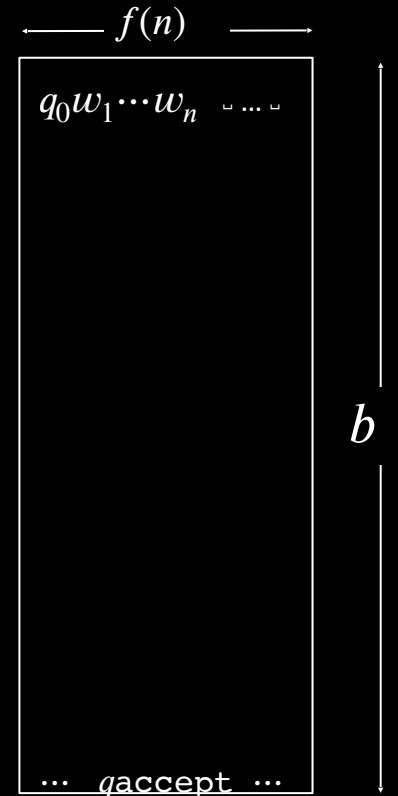
Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

1. If $b = 1$, check directly by using N 's program and answer accordingly.



PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

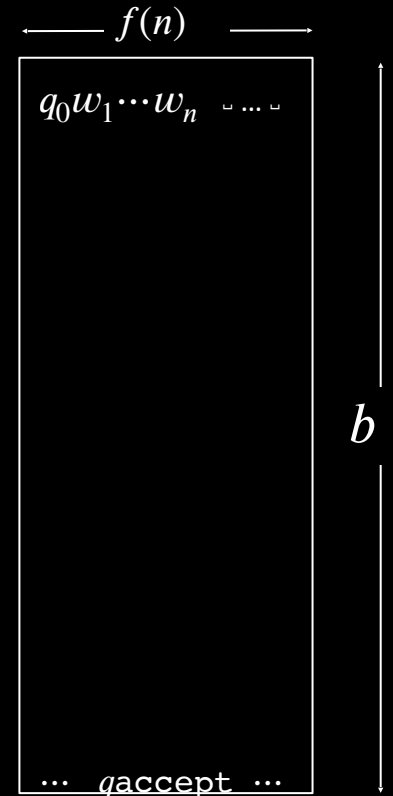
Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

1. If $b = 1$, check directly by using N 's program and answer accordingly.
2. If $b > 1$, repeat for all configurations c_{mid} that use $f(n)$ space.



PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

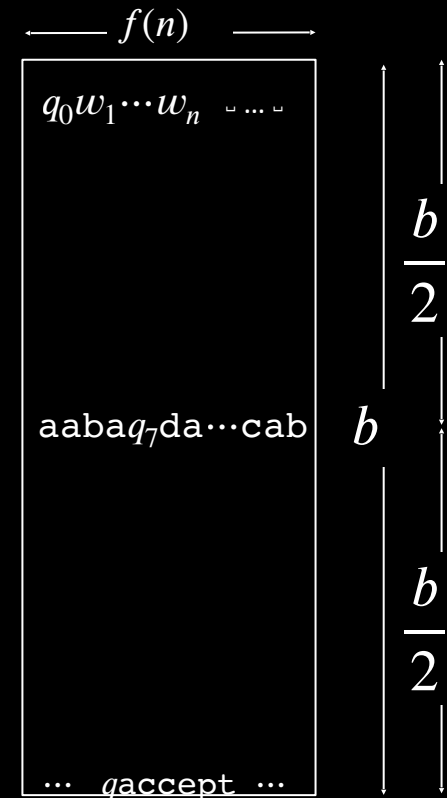
Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

1. If $b = 1$, check directly by using N 's program and answer accordingly.
2. If $b > 1$, repeat for all configurations c_{mid} that use $f(n)$ space.



PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

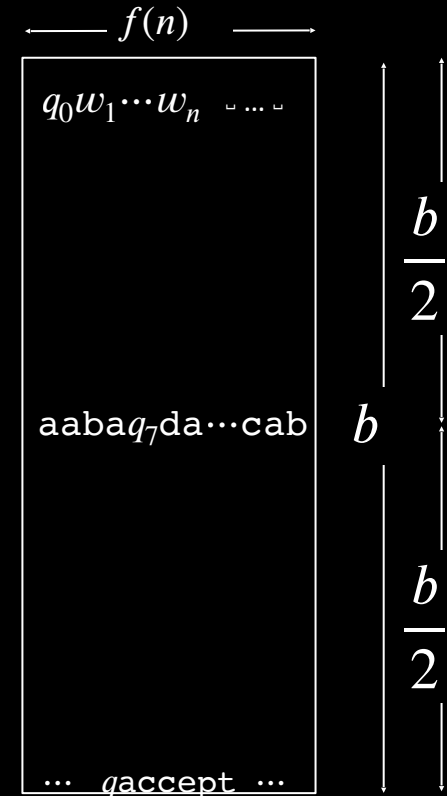
Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

1. If $b = 1$, check directly by using N 's program and answer accordingly.
2. If $b > 1$, repeat for all configurations c_{mid} that use $f(n)$ space.
3. Recursively test $c_i \xrightarrow{b/2} c_{\text{mid}}$ and $c_{\text{mid}} \xrightarrow{b/2} c_j$



PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

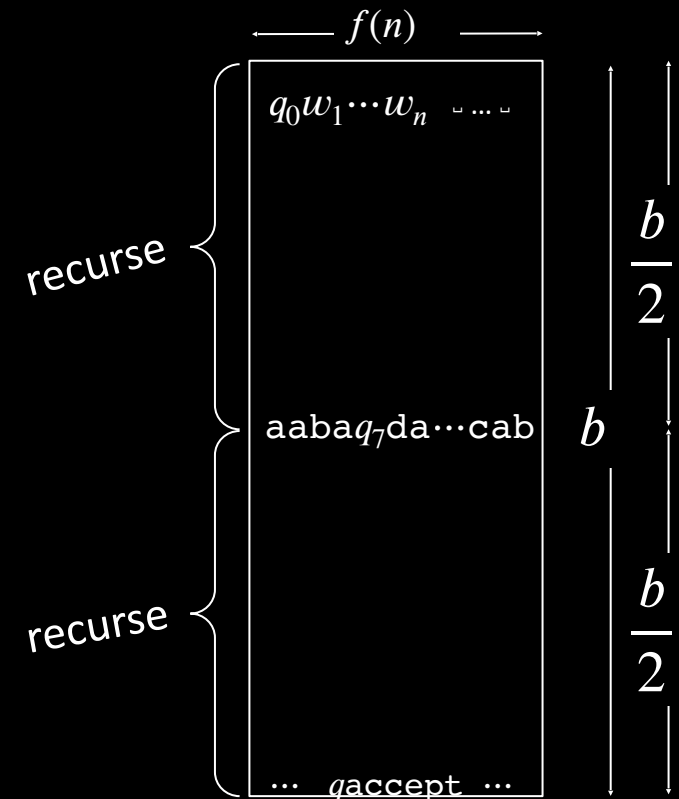
Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

1. If $b = 1$, check directly by using N 's program and answer accordingly.
2. If $b > 1$, repeat for all configurations c_{mid} that use $f(n)$ space.
3. Recursively test $c_i \xrightarrow{b/2} c_{\text{mid}}$ and $c_{\text{mid}} \xrightarrow{b/2} c_j$



PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

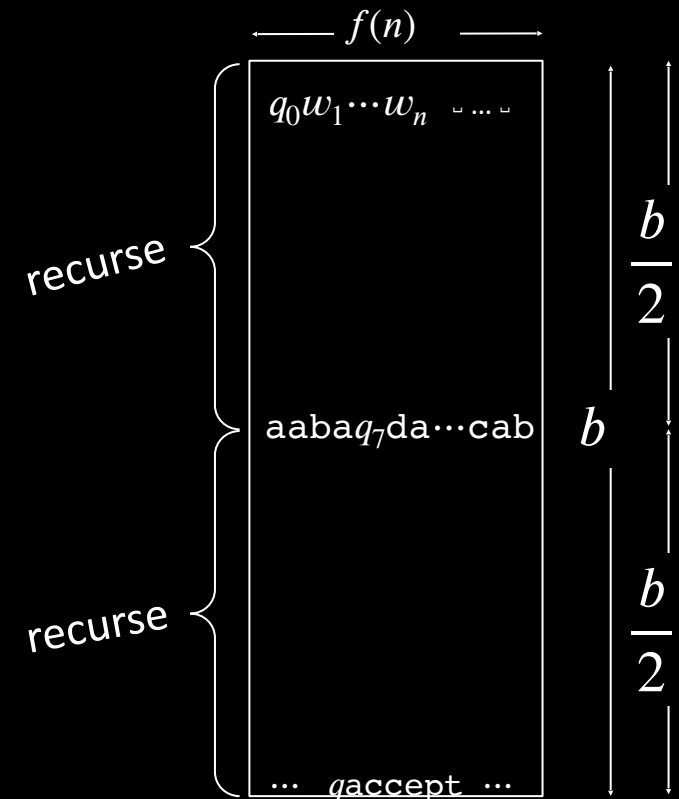
Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

1. If $b = 1$, check directly by using N 's program and answer accordingly.
2. If $b > 1$, repeat for all configurations c_{mid} that use $f(n)$ space.
3. Recursively test $c_i \xrightarrow{b/2} c_{\text{mid}}$ and $c_{\text{mid}} \xrightarrow{b/2} c_j$
4. If both are true, *accept*. If not, continue.



PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

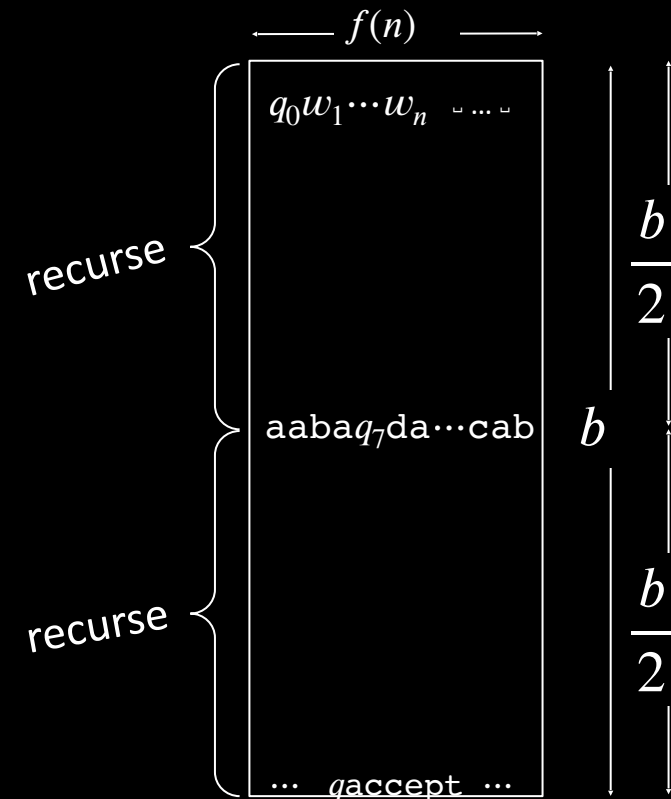
Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

1. If $b = 1$, check directly by using N 's program and answer accordingly.
2. If $b > 1$, repeat for all configurations c_{mid} that use $f(n)$ space.
3. Recursively test $c_i \xrightarrow{b/2} c_{\text{mid}}$ and $c_{\text{mid}} \xrightarrow{b/2} c_j$
4. If both are true, *accept*. If not, continue.
5. *Reject* if haven't yet accepted."



PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

Proof: Convert NTM N to equivalent TM M , only squaring the space used.

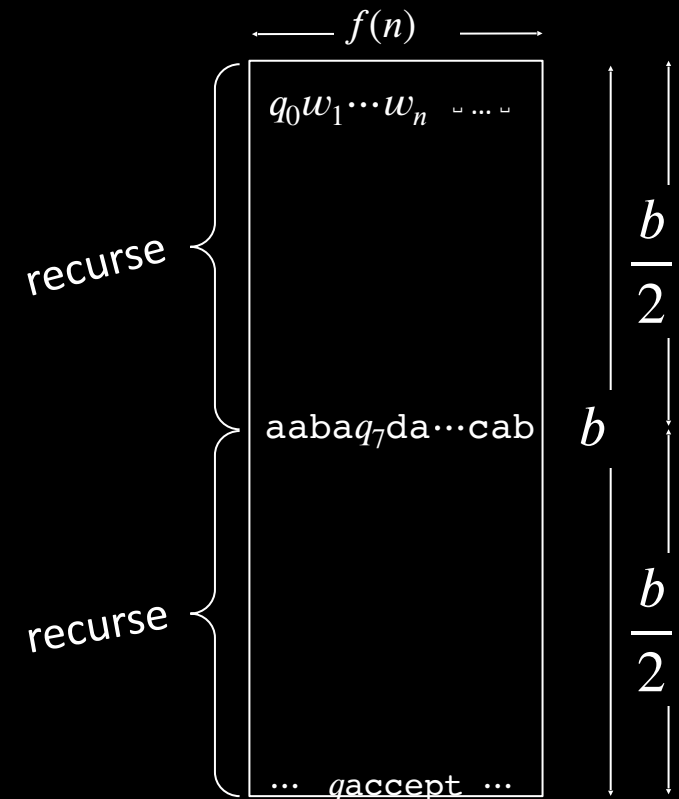
For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

1. If $b = 1$, check directly by using N 's program and answer accordingly.
2. If $b > 1$, repeat for all configurations c_{mid} that use $f(n)$ space.
3. Recursively test $c_i \xrightarrow{b/2} c_{\text{mid}}$ and $c_{\text{mid}} \xrightarrow{b/2} c_j$
4. If both are true, *accept*. If not, continue.
5. *Reject* if haven't yet accepted."

Test if N accepts w by testing $c_{\text{start}} \xrightarrow{t} c_{\text{accept}}$ where t = number of configurations



PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

Proof: Convert NTM N to equivalent TM M , only squaring the space used.

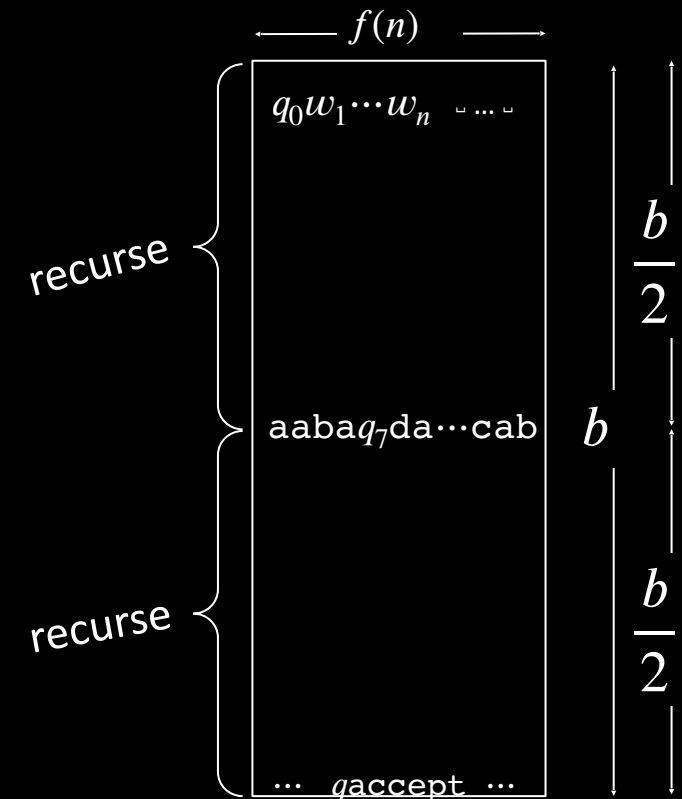
For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

1. If $b = 1$, check directly by using N 's program and answer accordingly.
2. If $b > 1$, repeat for all configurations c_{mid} that use $f(n)$ space.
3. Recursively test $c_i \xrightarrow{b/2} c_{\text{mid}}$ and $c_{\text{mid}} \xrightarrow{b/2} c_j$
4. If both are true, *accept*. If not, continue.
5. *Reject* if haven't yet accepted."

Test if N accepts w by testing $c_{\text{start}} \xrightarrow{t} c_{\text{accept}}$ where t = number of configurations
 $= |Q| \times f(n) \times d^{f(n)}$



PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

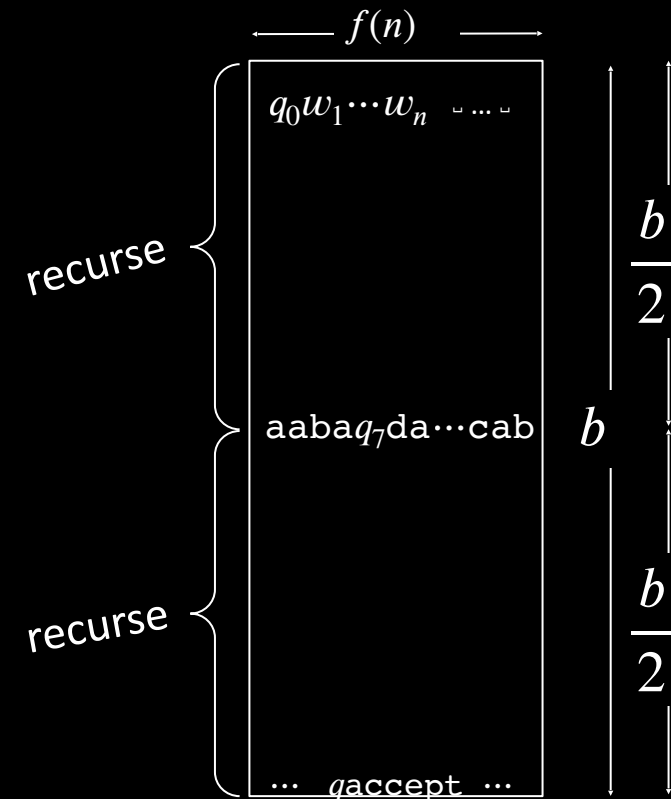
1. If $b = 1$, check directly by using N 's program and answer accordingly.
2. If $b > 1$, repeat for all configurations c_{mid} that use $f(n)$ space.
3. Recursively test $c_i \xrightarrow{b/2} c_{\text{mid}}$ and $c_{\text{mid}} \xrightarrow{b/2} c_j$
4. If both are true, *accept*. If not, continue.
5. *Reject* if haven't yet accepted."

Test if N accepts w by testing $c_{\text{start}} \xrightarrow{t} c_{\text{accept}}$ where t = number of configurations

$$= |Q| \times f(n) \times d^{f(n)}$$

Each recursion level stores 1 config = $O(f(n))$ space.

Number of levels = $\log t = O(f(n))$. Total $O(f^2(n))$ space.



PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \geq n$, $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$

Proof: Convert NTM N to equivalent TM M , only squaring the space used.

For configurations c_i and c_j of N , write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

M = "On input c_i, c_j, b [goal is to check $c_i \xrightarrow{b} c_j$]

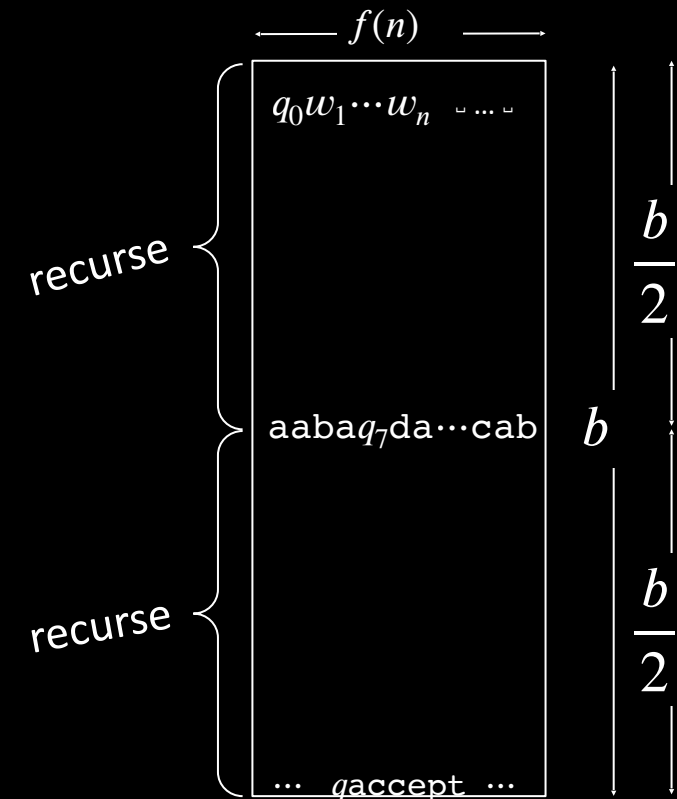
1. If $b = 1$, check directly by using N 's program and answer accordingly.
2. If $b > 1$, repeat for all configurations c_{mid} that use $f(n)$ space.
3. Recursively test $c_i \xrightarrow{b/2} c_{\text{mid}}$ and $c_{\text{mid}} \xrightarrow{b/2} c_j$
4. If both are true, *accept*. If not, continue.
5. *Reject* if haven't yet accepted."

Test if N accepts w by testing $c_{\text{start}} \xrightarrow{t} c_{\text{accept}}$ where t = number of configurations

$$= |Q| \times f(n) \times d^{f(n)}$$

Each recursion level stores 1 config = $O(f(n))$ space.

Number of levels = $\log t = O(f(n))$. Total $O(f^2(n))$ space.



PSPACE-completeness

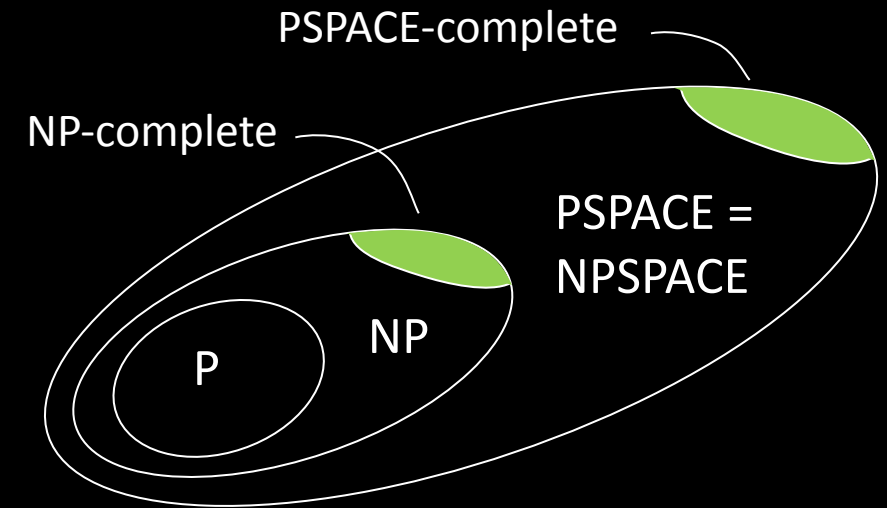
Defn: B is PSPACE-complete if

- 1) $B \in \text{PSPACE}$
- 2) For all $A \in \text{PSPACE}$, $A \leq_p B$

PSPACE-completeness

Defn: B is PSPACE-complete if

- 1) $B \in \text{PSPACE}$
- 2) For all $A \in \text{PSPACE}$, $A \leq_p B$



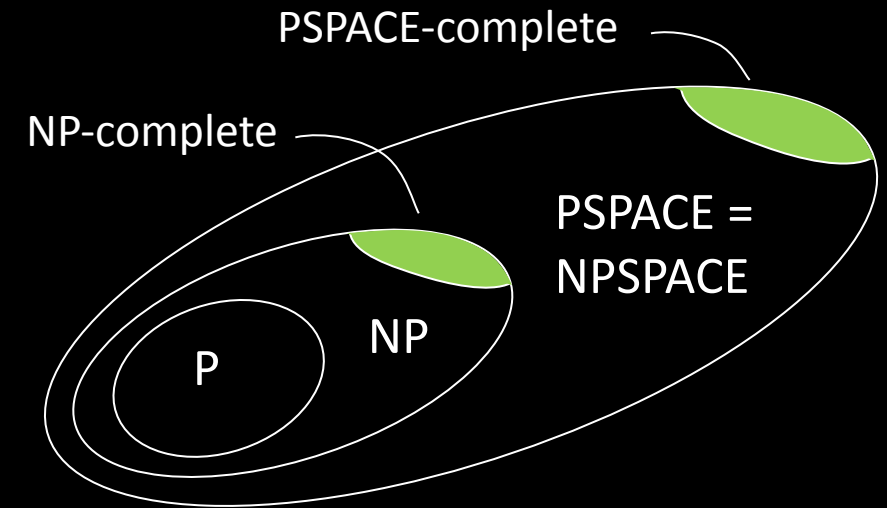
Think of complete problems as the “hardest” in their associated class.

PSPACE-completeness

Defn: B is PSPACE-complete if

- 1) $B \in \text{PSPACE}$
- 2) For all $A \in \text{PSPACE}$, $A \leq_p B$

If B is PSPACE-complete and $B \in P$ then $P = \text{PSPACE}$.



Think of complete problems as the “hardest” in their associated class.

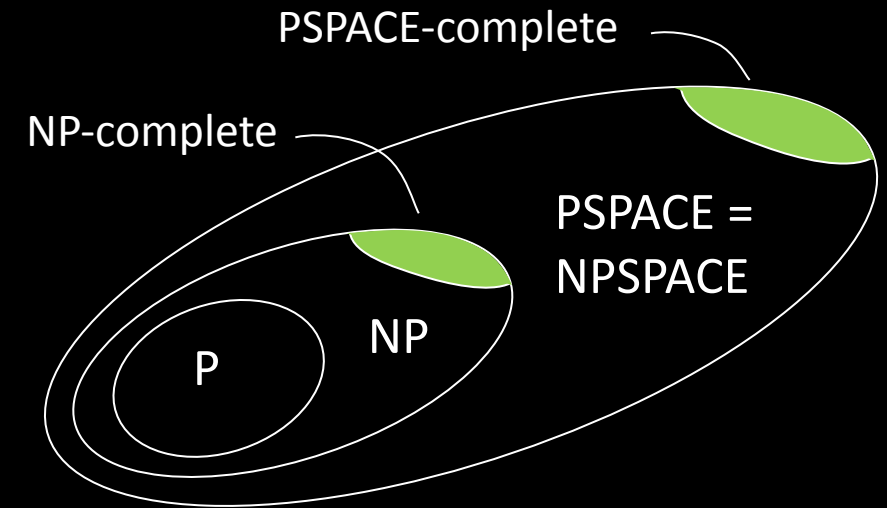
PSPACE-completeness

Defn: B is PSPACE-complete if

- 1) $B \in \text{PSPACE}$
- 2) For all $A \in \text{PSPACE}$, $A \leq_p B$

If B is PSPACE-complete and $B \in P$ then $P = \text{PSPACE}$.

Why \leq_p and not \leq_{PSPACE} when defining PSPACE-complete?



Think of complete problems as the “hardest” in their associated class.

PSPACE-completeness

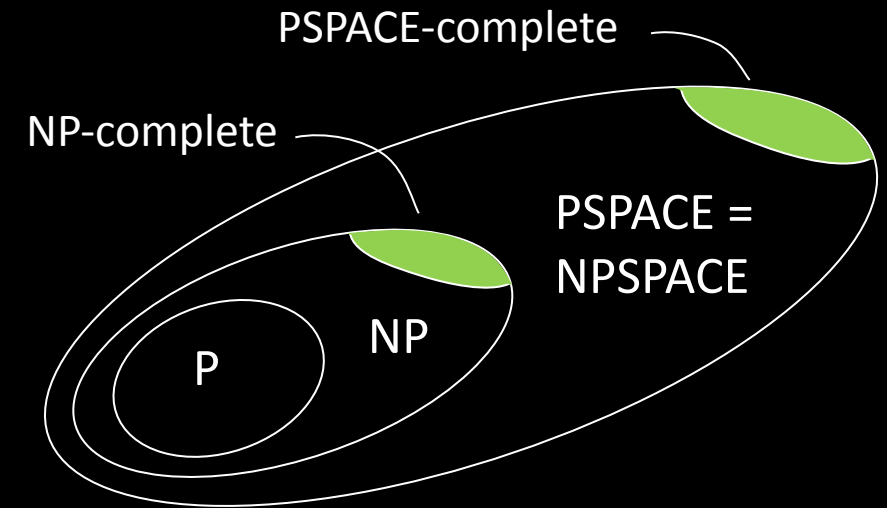
Defn: B is PSPACE-complete if

- 1) $B \in \text{PSPACE}$
- 2) For all $A \in \text{PSPACE}$, $A \leq_p B$

If B is PSPACE-complete and $B \in P$ then $P = \text{PSPACE}$.

Why \leq_p and not \leq_{PSPACE} when defining PSPACE-complete?

- Reductions should be “weaker” than the class. Otherwise all problems in the class would be reducible to each other, and then all problems in the class would be complete.



Think of complete problems as the “hardest” in their associated class.

PSPACE-completeness

Defn: B is PSPACE-complete if

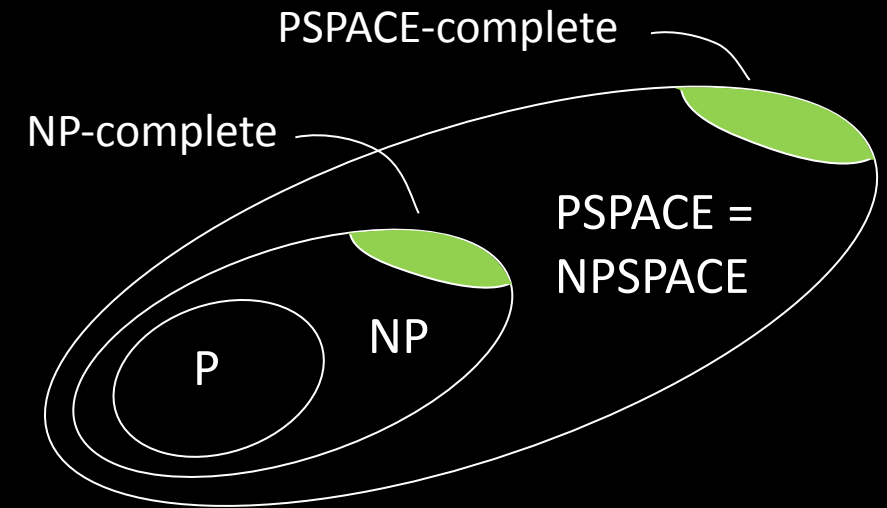
- 1) $B \in \text{PSPACE}$
- 2) For all $A \in \text{PSPACE}$, $A \leq_p B$

If B is PSPACE-complete and $B \in P$ then $P = \text{PSPACE}$.

Why \leq_p and not \leq_{PSPACE} when defining PSPACE-complete?

- Reductions should be “weaker” than the class. Otherwise all problems in the class would be reducible to each other, and then all problems in the class would be complete.

Theorem: $TQBF$ is PSPACE-complete



Think of complete problems as the “hardest” in their associated class.

PSPACE-completeness

Defn: B is PSPACE-complete if

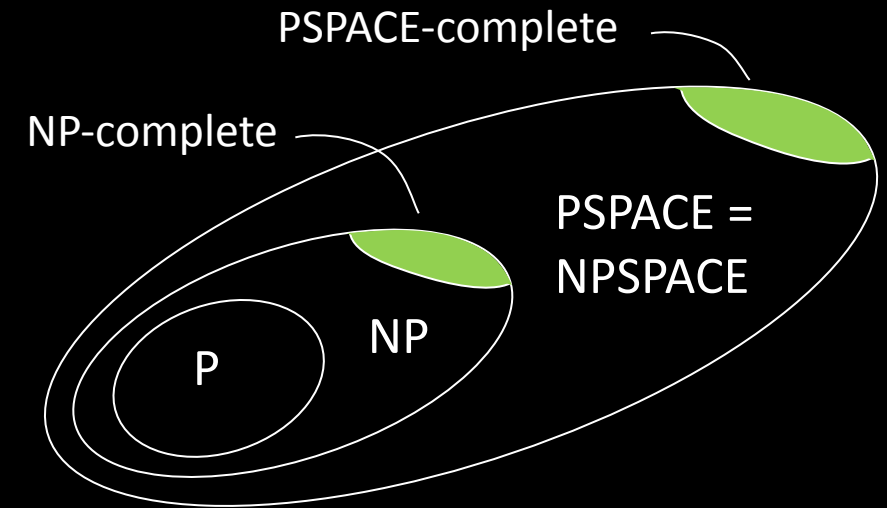
- 1) $B \in \text{PSPACE}$
- 2) For all $A \in \text{PSPACE}$, $A \leq_p B$

If B is PSPACE-complete and $B \in P$ then $P = \text{PSPACE}$.

Why \leq_p and not \leq_{PSPACE} when defining PSPACE-complete?

- Reductions should be “weaker” than the class. Otherwise all problems in the class would be reducible to each other, and then all problems in the class would be complete.

Theorem: $TQBF$ is PSPACE-complete



Think of complete problems as the “hardest” in their associated class.

PSPACE-completeness

Defn: B is PSPACE-complete if

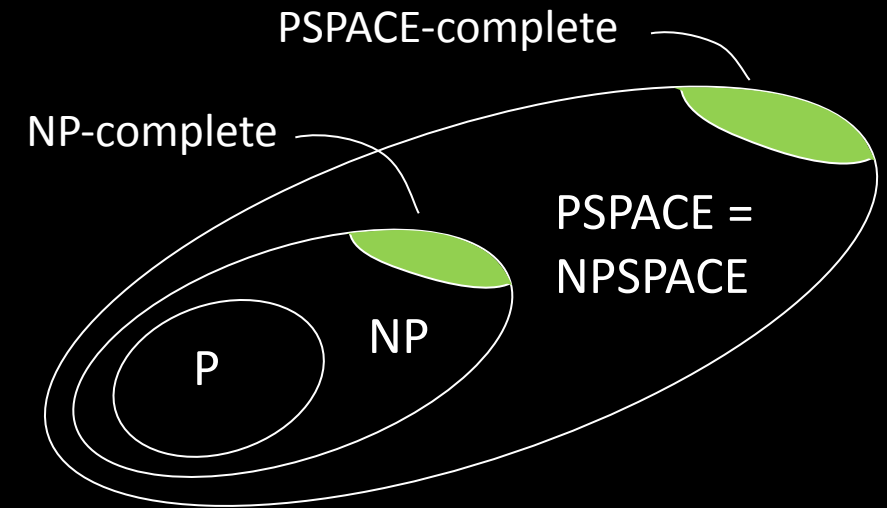
- 1) $B \in \text{PSPACE}$
- 2) For all $A \in \text{PSPACE}$, $A \leq_p B$

If B is PSPACE-complete and $B \in P$ then $P = \text{PSPACE}$.

Check-in 18.1

Knowing that $TQBF$ is PSPACE-complete, what can we conclude if $TQBF \in \text{NP}$? Check all that apply.

- (a) $P = \text{PSPACE}$
- (b) $\text{NP} = \text{PSPACE}$
- (c) $P = \text{NP}$
- (d) $\text{NP} = \text{coNP}$



Think of complete problems as the “hardest” in their associated class.

$TQBF$ is PSPACE-complete

Recall: $TQBF = \{ \langle \phi \rangle \mid \phi \text{ is a QBF that is TRUE} \}$

Examples: $\phi_1 = \forall x \exists y \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \in TQBF$ [TRUE]

$\phi_2 = \exists y \forall x \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \notin TQBF$ [FALSE]

$TQBF$ is PSPACE-complete

Recall: $TQBF = \{ \langle \phi \rangle \mid \phi \text{ is a QBF that is TRUE} \}$

Examples: $\phi_1 = \forall x \exists y \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \in TQBF$ [TRUE]

$\phi_2 = \exists y \forall x \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \notin TQBF$ [FALSE]

Theorem: $TQBF$ is PSPACE-complete

$TQBF$ is PSPACE-complete

Recall: $TQBF = \{ \langle \phi \rangle \mid \phi \text{ is a QBF that is TRUE} \}$

Examples: $\phi_1 = \forall x \exists y \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \in TQBF$ [TRUE]

$\phi_2 = \exists y \forall x \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \notin TQBF$ [FALSE]

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in \text{PSPACE}$ ✓

$TQBF$ is PSPACE-complete

Recall: $TQBF = \{ \langle \phi \rangle \mid \phi \text{ is a QBF that is TRUE} \}$

Examples: $\phi_1 = \forall x \exists y \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \in TQBF$ [TRUE]

$\phi_2 = \exists y \forall x \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \notin TQBF$ [FALSE]

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in \text{PSPACE}$ ✓

2) For all $A \in \text{PSPACE}$, $A \leq_p TQBF$

$TQBF$ is PSPACE-complete

Recall: $TQBF = \{ \langle \phi \rangle \mid \phi \text{ is a QBF that is TRUE} \}$

Examples: $\phi_1 = \forall x \exists y \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \in TQBF$ [TRUE]

$\phi_2 = \exists y \forall x \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \notin TQBF$ [FALSE]

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in \text{PSPACE}$ ✓

2) For all $A \in \text{PSPACE}$, $A \leq_p TQBF$

Let $A \in \text{PSPACE}$ be decided by TM M in space n^k .

Give a polynomial-time reduction f mapping A to $TQBF$.

$TQBF$ is PSPACE-complete

Recall: $TQBF = \{ \langle \phi \rangle \mid \phi \text{ is a QBF that is TRUE} \}$

Examples: $\phi_1 = \forall x \exists y \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \in TQBF$ [TRUE]

$\phi_2 = \exists y \forall x \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \notin TQBF$ [FALSE]

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in \text{PSPACE}$ ✓

2) For all $A \in \text{PSPACE}$, $A \leq_p TQBF$

Let $A \in \text{PSPACE}$ be decided by TM M in space n^k .

Give a polynomial-time reduction f mapping A to $TQBF$.

$f: \Sigma^* \rightarrow \text{QBFs}$

$f(w) = \langle \phi_{M,w} \rangle$

$w \in A$ iff $\phi_{M,w}$ is TRUE

$TQBF$ is PSPACE-complete

Recall: $TQBF = \{ \langle \phi \rangle \mid \phi \text{ is a QBF that is TRUE} \}$

Examples: $\phi_1 = \forall x \exists y \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \in TQBF$ [TRUE]

$\phi_2 = \exists y \forall x \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \notin TQBF$ [FALSE]

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in \text{PSPACE}$ ✓

2) For all $A \in \text{PSPACE}$, $A \leq_p TQBF$

Let $A \in \text{PSPACE}$ be decided by TM M in space n^k .

Give a polynomial-time reduction f mapping A to $TQBF$.

$$f: \Sigma^* \rightarrow \text{QBFs}$$

$$f(w) = \langle \phi_{M,w} \rangle$$

$$w \in A \text{ iff } \phi_{M,w} \text{ is TRUE}$$

Plan: Design $\phi_{M,w}$ to “say” M accepts w . $\phi_{M,w}$ simulates M on w .

$TQBF$ is PSPACE-complete

Recall: $TQBF = \{ \langle \phi \rangle \mid \phi \text{ is a QBF that is TRUE} \}$

Examples: $\phi_1 = \forall x \exists y \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \in TQBF$ [TRUE]

$\phi_2 = \exists y \forall x \left[(x \vee y) \wedge (\bar{x} \vee \bar{y}) \right] \notin TQBF$ [FALSE]

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in \text{PSPACE}$ ✓

2) For all $A \in \text{PSPACE}$, $A \leq_p TQBF$

Let $A \in \text{PSPACE}$ be decided by TM M in space n^k .

Give a polynomial-time reduction f mapping A to $TQBF$.

$$f: \Sigma^* \rightarrow \text{QBFs}$$

$$f(w) = \langle \phi_{M,w} \rangle$$

$$w \in A \text{ iff } \phi_{M,w} \text{ is TRUE}$$

Plan: Design $\phi_{M,w}$ to “say” M accepts w . $\phi_{M,w}$ simulates M on w .

Constructing $\phi_{M,w}$: 1st try

Tableau for M on w

[illegible]

Recall: A tableau for M on w represents a computation history for M on w when M accepts w . Rows of that tableau are configurations.

Constructing $\phi_{M,w}$: 1st try

Tableau for M on w

			<div> <div></div> <div>...</div> <div></div> </div>
a			

Recall: A tableau for M on w represents
a computation history for M on w
when M accepts w .

Rows of that tableau are configurations.

M runs in space n^k , its tableau has:

Constructing $\phi_{M,w}$: 1st try

Tableau for M on w

			<div> <div></div> <div>...</div> <div></div> </div>
a			

Recall: A tableau for M on w represents
a computation history for M on w
when M accepts w .

Rows of that tableau are configurations.

M runs in space n^k , its tableau has:

- n^k columns (max size of a configuration)

Constructing $\phi_{M,w}$: 1st try

Tableau for M on w

[illegible]

Recall: A tableau for M on w represents
a computation history for M on w
when M accepts w .

Rows of that tableau are configurations.

M runs in space n^k , its tableau has:

- n^k columns (max size of a configuration)

Constructing $\phi_{M,w}$: 1st try

Tableau for M on w

[illegible]

Recall: A tableau for M on w represents a computation history for M on w when M accepts w .

Rows of that tableau are configurations.

M runs in space n^k , its tableau has:

- n^k columns (max size of a configuration)
- $d^{(n^k)}$ rows (max number of steps)

Constructing $\phi_{M,w}$: 1st try

Tableau for M on w

The diagram illustrates a 3D tensor $d^{(n^k)}$ with three dimensions, each of size n^k . The tensor is represented as a cube with a grid of cells on its visible faces. The top face is labeled n^k and has a grid of n^k columns and n^k rows. The front face is labeled $d^{(n^k)}$ and has a grid of n^k columns and n^k rows. The right face is labeled n^k and has a grid of n^k columns and n^k rows. The cells are arranged in a 3D grid, with the top face cells being white, the front face cells being light blue, and the right face cells being light green. The cells are arranged in a 3D grid, with the top face cells being white, the front face cells being light blue, and the right face cells being light green. The cells are arranged in a 3D grid, with the top face cells being white, the front face cells being light blue, and the right face cells being light green.

Recall: A tableau for M on w represents a computation history for M on w when M accepts w .

Rows of that tableau are configurations.

M runs in space n^k , its tableau has:

- n^k columns (max size of a configuration)
- $d^{(n^k)}$ rows (max number of steps)

Constructing $\phi_{M,w}$: 1st try

Tableau for M on w

Diagram illustrating a 3D tensor $d^{(n^k)}$ with dimensions n^k , n^k , and n^k . The tensor is represented as a cube with three axes labeled n^k . The top face is a grid with 3 columns and 3 rows. The left face is a grid with 3 columns and 3 rows. The front face is a grid with 3 columns and 3 rows. The tensor is labeled $d^{(n^k)}$ on the left.

Recall: A tableau for M on w represents a computation history for M on w when M accepts w .

Rows of that tableau are configurations.

M runs in space n^k , its tableau has:

- n^k columns (max size of a configuration)
- $d^{(n^k)}$ rows (max number of steps)

Constructing $\phi_{M,w}$. Try Cook-Levin method.

Constructing $\phi_{M,w}$: 1st try

Tableau for M on w

Diagram illustrating a 3D tensor $d^{(n^k)}$ with dimensions n^k , n^k , and n^k . The tensor is represented as a 3D box with three axes. The top horizontal axis is labeled n^k . The vertical axis is labeled $d^{(n^k)}$. The depth axis is labeled n^k . The tensor is divided into a 2x2x2 grid of smaller blocks.

Recall: A tableau for M on w represents a computation history for M on w when M accepts w .

Rows of that tableau are configurations.

M runs in space n^k , its tableau has:

- n^k columns (max size of a configuration)
- $d^{(n^k)}$ rows (max number of steps)

Constructing $\phi_{M,w}$. Try Cook-Levin method.

Then $\phi_{M,w}$ will be as big as tableau.

Constructing $\phi_{M,w}$: 1st try

Tableau for M on w

The diagram illustrates a 3D tensor $d^{(n^k)}$. It consists of a large rectangular prism. The top horizontal edge is labeled n^k . The left vertical edge is labeled $d^{(n^k)}$. The depth edge, receding from the top-left corner, is also labeled n^k . The prism is divided into a grid of smaller rectangular cells by lines parallel to its edges. The front face shows a grid of cells, with the top-left cell containing the letter 'a'. The top face shows a grid of cells, with the top-right cell containing an ellipsis '...'.

Recall: A tableau for M on w represents a computation history for M on w when M accepts w .

Rows of that tableau are configurations.

M runs in space n^k , its tableau has:

- n^k columns (max size of a configuration)
- $d^{(n^k)}$ rows (max number of steps)

Constructing $\phi_{M,w}$. Try Cook-Levin method.

Then $\phi_{M,w}$ will be as big as tableau.

But that is exponential: $n^k \times d^{(n^k)}$.

Constructing $\phi_{M,w}$: 1st try

Tableau for M on w

Diagram illustrating a 3D tensor $d^{(n^k)}$ with dimensions n^k , n^k , and n^k . The tensor is represented as a 3D box with three axes. The top horizontal axis is labeled n^k . The vertical axis is labeled $d^{(n^k)}$. The depth axis is labeled n^k . The tensor is divided into a 3x3x3 grid of smaller boxes.

Recall: A tableau for M on w represents a computation history for M on w when M accepts w .

Rows of that tableau are configurations.

M runs in space n^k , its tableau has:

- n^k columns (max size of a configuration)
- $d^{(n^k)}$ rows (max number of steps)

Constructing $\phi_{M,w}$. Try Cook-Levin method.

Then $\phi_{M,w}$ will be as big as tableau.

But that is exponential: $n^k \times d^{(n^k)}$.

Too big! ☹️

Constructing $\phi_{M,w}$: 1st try

Tableau for M on w

[illegible]

Recall: A tableau for M on w represents a computation history for M on w when M accepts w .

Rows of that tableau are configurations.

M runs in space n^k , its tableau has:

- n^k columns (max size of a configuration)
- $d^{(n^k)}$ rows (max number of steps)

Constructing $\phi_{M,w}$. Try Cook-Levin method.

Then $\phi_{M,w}$ will be as big as tableau.

But that is exponential: $n^k \times d^{(n^k)}$.

Too big! ☹️

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

The diagram illustrates the construction of a sequence of partitions $d^{(n^k)}$ for $k=1, 2, 3, 4$. The partitions are shown as nested structures, with the top partition $d^{(n^4)}$ being the most complex and the bottom partition $d^{(n^1)}$ being the simplest. The partitions are constructed by adding new elements to the previous partition, as indicated by the arrows and the labels n^k .

$$d^{(n^k)}$$

hide →

$$v^k)$$
[illegible]

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

The diagram illustrates a 3D tensor $d^{(n^k)}$ with three dimensions, each of size n^k . The tensor is represented as a large cube. The top face is divided into a 3×3 grid of smaller cubes. The front face is divided into a 3×3 grid of smaller cubes. The right face is divided into a 3×3 grid of smaller cubes. The tensor is labeled $d^{(n^k)}$ on the left.

$$d^{(n^k)}$$

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$d^{(n^k)}$				⌊ ... ⌊
	a			

$$d^{(n^k)}$$

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$$\phi_{c_i, c_j, b} = \exists c_{\text{mid}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

[illegible]
$$d^{(n^k)}$$

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

$d^{(n^k)}$

$\xrightarrow{\quad n^k \quad}$			
			⋮ ⋮ ⋮
a			

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$$\phi_{c_i, c_j, b} = \underbrace{\exists c_{\text{mid}}}_{\substack{\exists x_1, x_2, \dots, c_l \\ \text{as in Cook-Levin}}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

$d^{(n^k)}$

n^k			
			⋮ ⋮ ⋮
a			

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$$\phi_{c_i, c_j, b} = \underbrace{\exists c_{\text{mid}}}_{\substack{\exists x_1, x_2, \dots, x_l \\ \text{as in Cook-Levin}}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

$$\exists c_{\text{mid}} \left[\phi_{, , b/4} \wedge \phi_{, , b/4} \right]$$

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

$d^{(n^k)}$

n^k			
a			

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$$\phi_{c_i, c_j, b} = \underbrace{\exists c_{\text{mid}}}_{\substack{\exists x_1, x_2, \dots, x_l \\ \text{as in Cook-Levin}}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

$\swarrow \quad \searrow$
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right]$

$\swarrow \quad \searrow$
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right]$

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

$d^{(n^k)}$

n^k			
a			

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$$\phi_{c_i, c_j, b} = \underbrace{\exists c_{\text{mid}}}_{\substack{\exists x_1, x_2, \dots, x_l \\ \text{as in Cook-Levin}}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

$\swarrow \quad \searrow$
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right]$

$\swarrow \quad \searrow$
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right]$
 $\swarrow \quad \searrow$
 $\exists c_{\text{mid}} [\phi_{\cdot, \cdot, b/8} \dots]$

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

$d^{(n^k)}$

n^k			

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$$\phi_{c_i, c_j, b} = \underbrace{\exists c_{\text{mid}}}_{\substack{\exists x_1, x_2, \dots, x_l \\ \text{as in Cook-Levin}}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

$\swarrow \quad \searrow$
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right]$
 \vdots

$\swarrow \quad \searrow$
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right]$
 \vdots
 $\swarrow \quad \searrow$
 $\exists c_{\text{mid}} [\phi_{\cdot, \cdot, b/8} \cdots]$

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

$d^{(n^k)}$

n^k			

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$$\phi_{c_i, c_j, b} = \underbrace{\exists c_{\text{mid}}}_{\substack{\exists x_1, x_2, \dots, c_l \\ \text{as in Cook-Levin}}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

$\downarrow \qquad \downarrow$
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right]$
 \vdots
 $\phi_{\cdot, \cdot, 1}$ defined as in Cook-Levin

$\downarrow \qquad \downarrow$
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right]$
 \vdots
 $\exists c_{\text{mid}} [\phi_{\cdot, \cdot, b/8} \dots]$

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

$d^{(n^k)}$

n^k			
			... \vdash
a			

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$$\phi_{c_i, c_j, b} = \underbrace{\exists c_{\text{mid}}}_{\substack{\exists x_1, x_2, \dots, c_l \\ \text{as in Cook-Levin}}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

$\phi_{, , b/4} \wedge \phi_{, , b/4}$
 \vdots
 $\phi_{, , 1}$ defined as in Cook-Levin

$\phi_{, , b/4} \wedge \phi_{, , b/4}$
 \vdots
 $\exists c_{\text{mid}} [\phi_{, , b/8} \dots]$

$$\phi_{M,w} = \phi_{c_{\text{start}}, c_{\text{accept}}, t}$$

$$t = d^{(n^k)}$$

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

$d^{(n^k)}$

n^k			
			...
a			

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$$\phi_{c_i, c_j, b} = \underbrace{\exists c_{\text{mid}}}_{\substack{\exists x_1, x_2, \dots, c_l \\ \text{as in Cook-Levin}}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

$\downarrow \quad \downarrow$
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right]$
 \vdots

$\downarrow \quad \downarrow$
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right]$
 \vdots
 $\downarrow \quad \downarrow$
 $\exists c_{\text{mid}} [\phi_{\cdot, \cdot, b/8} \cdots]$

$\phi_{\cdot, \cdot, 1}$ defined as in Cook-Levin

$$\phi_{M,w} = \phi_{c_{\text{start}}, c_{\text{accept}}, t}$$

$$t = d^{(n^k)}$$

Size analysis:
Each recursive level doubles number of QBFs.

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

$d^{(n^k)}$

n^k			
			...
a			

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$$\phi_{c_i, c_j, b} = \underbrace{\exists c_{\text{mid}}}_{\substack{\exists x_1, x_2, \dots, x_l \\ \text{as in Cook-Levin}}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

$\downarrow \quad \downarrow$
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right]$
 \vdots
 $\phi_{\cdot, \cdot, 1}$ defined as in Cook-Levin

$\downarrow \quad \downarrow$
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right]$
 \vdots
 $\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/8} \dots \right]$

$$\phi_{M,w} = \phi_{c_{\text{start}}, c_{\text{accept}}, t}$$

$$t = d^{(n^k)}$$

Size analysis:

Each recursive level doubles number of QBFs.
 Number of levels is $\log d^{(n^k)} = O(n^k)$.

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

$$n^k$$

a

□ • • • □

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

[illegible]

 $\phi_{\epsilon, \delta, 1}$ defined as in Cook-Levin

$$\phi_{M,w} = \phi_{c_{\text{start}}, c_{\text{accept}}, t}$$

$$t = d^{(n^k)}$$

Size analysis:

Each recursive level doubles number of QBFs.
Number of levels is $\log d^{(n^k)} = O(n^k)$.

→ Size is exponential. ☹️

Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w

For configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$$\begin{array}{c}
\phi_{c_i, c_j, b} = \underbrace{\exists c_{\text{mid}}}_{\substack{\exists x_1, x_2, \dots, c_l \\ \text{as in Cook-Levin}}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right] \\
\begin{array}{ccc}
\swarrow & & \searrow \\
\exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right] & & \exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right] \\
\vdots & & \vdots \\
\phi_{\cdot, \cdot, 1} \text{ defined as in Cook-Levin} & & \exists c_{\text{mid}} [\phi_{\cdot, \cdot, b/8} \dots]
\end{array}
\end{array}$$

$$\phi_{M,w} = \phi_{c_{\text{start}}, c_{\text{accept}}, t}$$

Size analysis:

Each recursive level doubles number of QBFs.
Number of levels is $\log d^{(n^k)} = O(n^k)$.

→ Size is exponential. ☹️

Check-in 18.2

Constructing $\phi_{M,w}$: 2nd try

· configs c_i and c_j construct $\phi_{c_i, c_j, b}$ which “says” $c_i \xrightarrow{b} c_j$ recursively.

$$\phi_{c_i, c_j, b} = \underbrace{\exists c_{\text{mid}}}_{\substack{\exists x_1, x_2, \dots, x_l \\ \text{as in Cook-Levin}}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

$$\begin{array}{c} \swarrow \quad \searrow \qquad \qquad \swarrow \quad \searrow \\ \exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right] \quad \exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/4} \wedge \phi_{\cdot, \cdot, b/4} \right] \\ \vdots \qquad \qquad \qquad \vdots \\ \exists c_{\text{mid}} \left[\phi_{\cdot, \cdot, b/8} \dots \right] \end{array}$$

$\phi_{\cdot, \cdot, 1}$ defined as in Cook-Levin

Check-in 18.2

Why shouldn't we be surprised that this construction fails?

- (a) We can't define a QBF by using recursion.
- (b) It doesn't use \forall anywhere.
- (c) We know that $TQBF \notin \text{P}$.

$$\phi_{M,w} = \phi_{c_{\text{start}}, c_{\text{accept}}, t}$$

$$t = d^{(n^k)}$$

Size analysis:

Each recursive level doubles number of QBFs.
Number of levels is $\log d^{(n^k)} = O(n^k)$.

→ Size is exponential. ☹

Check-in 18.2

Constructing $\phi_{M,w}$: 3rd try

$$\phi_{c_i, c_j, b} = \exists c_{\text{mid}} \left[\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2} \right]$$

Constructing $\phi_{M,w}$: 3rd try

$$\phi_{c_i, c_j, b} = \exists c_{\text{mid}} \left[\underbrace{\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2}}_{\text{}} \right]$$
$$\forall (c_g, c_h) \in \left\{ (c_i, c_{\text{mid}}), (c_{\text{mid}}, c_j) \right\} \left[\phi_{c_g, c_h, b/2} \right]$$

Constructing $\phi_{M,w}$: 3rd try

$$\phi_{c_i, c_j, b} = \exists c_{\text{mid}} \left[\underbrace{\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2}}_{\text{...}} \right]$$
$$\forall (c_g, c_h) \in \left\{ (c_i, c_{\text{mid}}), (c_{\text{mid}}, c_j) \right\} \left[\phi_{c_g, c_h, b/2} \right]$$
$$\vdots$$

Constructing $\phi_{M,w}$: 3rd try

$$\phi_{c_i, c_j, b} = \exists c_{\text{mid}} \left[\underbrace{\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2}} \right]$$

$$\forall (c_g, c_h) \in \left\{ (c_i, c_{\text{mid}}), (c_{\text{mid}}, c_j) \right\} \left[\phi_{c_g, c_h, b/2} \right]$$

$$\vdots$$

$\phi_{, , 1}$ defined as in Cook-Levin

Constructing $\phi_{M,w}$: 3rd try

$$\phi_{c_i, c_j, b} = \exists c_{\text{mid}} \left[\underbrace{\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2}} \right]$$

$$\forall (c_g, c_h) \in \left\{ (c_i, c_{\text{mid}}), (c_{\text{mid}}, c_j) \right\} \left[\phi_{c_g, c_h, b/2} \right]$$

$$\vdots$$

$\forall (x \in S) [\psi]$
 is equivalent to
 $\forall x [(x \in S) \rightarrow \psi]$

$\phi_{, , 1}$ defined as in Cook-Levin

Constructing $\phi_{M,w}$: 3rd try

$$\phi_{c_i, c_j, b} = \exists c_{\text{mid}} \left[\underbrace{\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2}} \right]$$

$$\forall (c_g, c_h) \in \left\{ (c_i, c_{\text{mid}}), (c_{\text{mid}}, c_j) \right\} \left[\phi_{c_g, c_h, b/2} \right]$$

$$\vdots$$

$\forall (x \in S) [\psi]$
is equivalent to
 $\forall x [(x \in S) \rightarrow \psi]$

$$\phi_{M,w} = \phi_{c_{\text{start}}, c_{\text{accept}}, t}$$

$$t = d(n^k)$$

$\phi_{, , 1}$ defined as in Cook-Levin

Constructing $\phi_{M,w}$: 3rd try

$$\phi_{c_i, c_j, b} = \exists c_{\text{mid}} \left[\underbrace{\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2}} \right]$$

$$\forall (c_g, c_h) \in \left\{ (c_i, c_{\text{mid}}), (c_{\text{mid}}, c_j) \right\} \left[\phi_{c_g, c_h, b/2} \right]$$

$$\vdots$$

$\forall (x \in S) [\psi]$
is equivalent to
 $\forall x [(x \in S) \rightarrow \psi]$

$$\phi_{M,w} = \phi_{c_{\text{start}}, c_{\text{accept}}, t}$$

$$t = d^{(n^k)}$$

$\phi_{, , 1}$ defined as in Cook-Levin

Size analysis:

Each recursive level adds $O(n^k)$ to the QBF.

Number of levels is $\log d^{(n^k)} = O(n^k)$.

→ Size is $O(n^k \times n^k) = O(n^{2k})$ ☺

Constructing $\phi_{M,w}$: 3rd try

$$\phi_{c_i, c_j, b} = \exists c_{\text{mid}} \left[\underbrace{\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2}} \right]$$

$$\forall (c_g, c_h) \in \left\{ (c_i, c_{\text{mid}}), (c_{\text{mid}}, c_j) \right\} \left[\phi_{c_g, c_h, b/2} \right]$$

$$\vdots$$

$\forall (x \in S) [\psi]$
is equivalent to
 $\forall x [(x \in S) \rightarrow \psi]$

$$\phi_{M,w} = \phi_{c_{\text{start}}, c_{\text{accept}}, t}$$

$$t = d^{(n^k)}$$

$\phi_{, , 1}$ defined as in Cook-Levin

Size analysis:

Each recursive level adds $O(n^k)$ to the QBF.

Number of levels is $\log d^{(n^k)} = O(n^k)$.

→ Size is $O(n^k \times n^k) = O(n^{2k})$ ☺

Constructing $\phi_{M,w}$: 3rd try

$$\phi_{c_i, c_j, b} = \exists c_{\text{mid}} \left[\underbrace{\phi_{c_i, c_{\text{mid}}, b/2} \wedge \phi_{c_{\text{mid}}, c_j, b/2}} \right]$$

$$\forall (c_g, c_h) \in \left\{ (c_i, c_{\text{mid}}), (c_{\text{mid}}, c_j) \right\} \left[\phi_{c_g, c_h, b/2} \right]$$

$$\vdots$$

$\forall (x \in S) [\psi]$
is equivalent to
 $\forall x [(x \in S) \rightarrow \psi]$

$$\phi_{M,w} = \phi_{c_{\text{start}}, c_{\text{accept}}, t}$$

$$t = d^{(n^k)}$$

$\phi_{, , 1}$ defined as in Cook-Levin

Size analysis:

Each recursive level adds $O(n^k)$ to the QBF.
Number of levels is $\log d^{(n^k)} = O(n^k)$.

→ Size is $O(n^k \times n^k) = O(n^{2k})$ ☺

Check-in 18.3

Would this construction still work if M were nondeterministic?

- (a) Yes.
- (b) No.

Check-in 18.3

Quick review of today

1. Space complexity
2. $\text{SPACE}(f(n))$, $\text{NSPACE}(f(n))$
3. PSPACE, NPSPACE
4. Relationship with TIME classes
5. $TQBF \in \text{PSPACE}$
6. $LADDER_{\text{DFA}} \in \text{NSPACE}(n)$
7. $LADDER_{\text{DFA}} \in \text{SPACE}(n^2)$
8. Savitch's Theorem: $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$
9. $TQBF$ is PSPACE-complete

Quick review of today

1. Space complexity
2. $\text{SPACE}(f(n))$, $\text{NSPACE}(f(n))$
3. PSPACE, NPSPACE
4. Relationship with TIME classes
5. $TQBF \in \text{PSPACE}$
6. $LADDER_{\text{DFA}} \in \text{NSPACE}(n)$
7. $LADDER_{\text{DFA}} \in \text{SPACE}(n^2)$
8. Savitch's Theorem: $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$
9. $TQBF$ is PSPACE-complete