بسم الله الرحمن الرحيم

نظریه علوم کامپیوتر

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

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

Review: SPACE Complexity

Defn: Let $f\colon \mathbb{N} \rightarrow \mathbb{N}$ 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 $f\colon \mathbb{N} \to \mathbb{N}$ 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 $f\colon \mathbb{N} \to \mathbb{N}$ 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.

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

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

Review: SPACE Complexity

Defn: Let $f\colon \mathbb{N} \to \mathbb{N}$ 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.

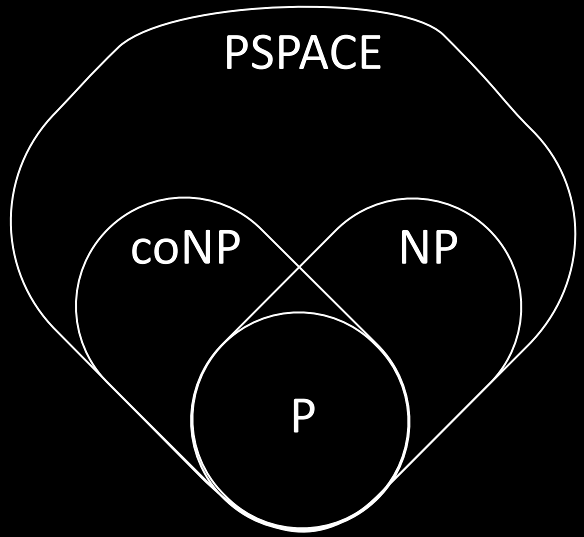
An NTM M runs in space $f(n)$ if all branches halt and each branch uses at most $f(n)$ tape cells on all inputs of length n.

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

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

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

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



Or possibly: P = NP = coNP = PSPACE

Review: SPACE Complexity

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

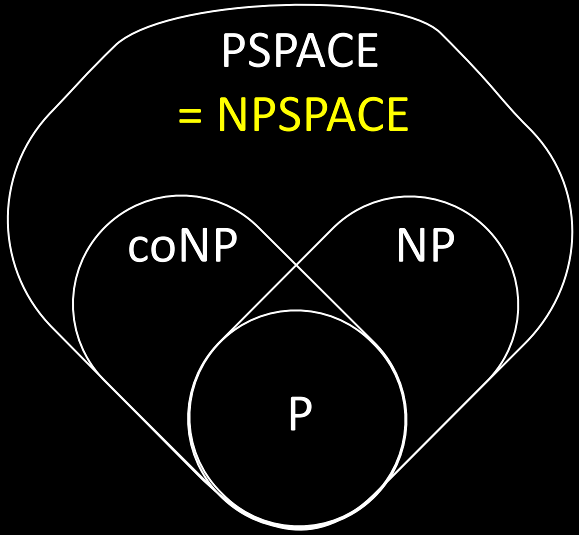
An NTM M <u>runs in space</u> $f(n)$ if all branches halt and each branch uses at most $f(n)$ tape cells on all inputs of length n.

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

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

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

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



Today: PSPACE = NPSPACE

Or possibly: P = NP = coNP = PSPACE

Review: SPACE Complexity

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

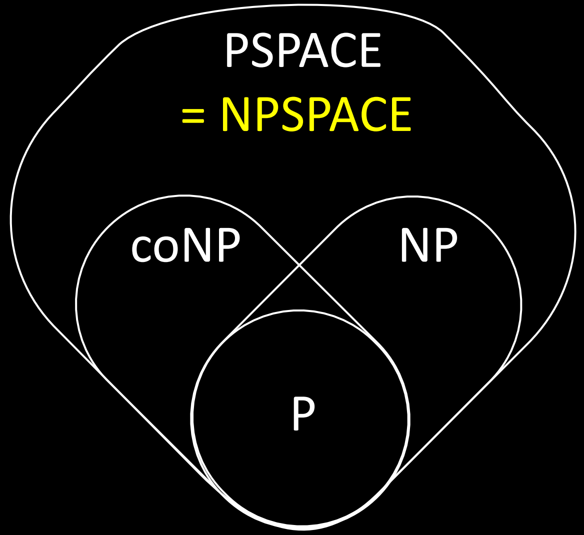
An NTM M <u>runs in space</u> $f(n)$ if all branches halt and each branch uses at most $f(n)$ tape cells on all inputs of length n.

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

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

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

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



Today: PSPACE = NPSPACE

Or possibly: P = NP = coNP = PSPACE

Example: Ladder Problem

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

Example: Ladder Problem

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

A <u>word ladder for English</u> 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 <u>ladder</u> is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A <u>word ladder for English</u> 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 <u>ladder</u> is a sequence of strings of a common length where consecutive strings differ in a single symbol.

A <u>word ladder for English</u> 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: $LADDER_{\mathbf{DFA}} = \{\langle B, u, v\rangle \mid B$ is a DFA and $L(B)$ contains a ladder $y_1,\ y_2,\ \ldots,\ y_k$ where $y_1 = u$ 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: $LADDER$**DFA** $= \left\{ \langle B, u, v \rangle \;\middle|\; B\right.$ is a DFA and $L(B)$ contains a ladder $y_1,\ y_2,\ \ldots,\ y_k$ where $y_1 = u$ and $y_k = v\}$.

Theorem: $LADDER$**DFA** $\in$ 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: $\mathit{LADDER}\textbf{DFA} = \{\langle B, u, v\rangle \mid B$ is a DFA and $L(B)$ contains a ladder $y_1,\ y_2,\ \ldots,\ y_k$ where $y_1 = u$ and $y_k = v\}$.

Theorem: $\mathit{LADDER}\text{DFA} \in$ NPSPACE

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER_{\text{DFA}} \in \text{NPSPACE}$

Theorem: $LADDER_{\text{DFA}} \in \text{NPSPACE}$

$LADDER$DFA $\in$ NPSPACE

Theorem: $LADDER$DFA $\in$ NPSPACE

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

$LADDER_{\mathrm{DFA}} \in$ NPSPACE

Theorem: $LADDER_{\mathrm{DFA}} \in$ NPSPACE

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

$\leq t$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER_{\text{DFA}} \in$ NPSPACE

Theorem: $LADDER_{\text{DFA}} \in$ NPSPACE

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

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

$\leq t$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER$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 $\langle B, u, v \rangle$

$\leq t$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER_{\text{DFA}} \in$ NPSPACE

Theorem: $LADDER_{\text{DFA}} \in$ 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 $m = |u|$.

$\leq t$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER_{\text{DFA}} \in$ NPSPACE

Theorem: $LADDER_{\text{DFA}} \in$ 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 $t = |\Sigma|^m$.

$\le t$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER_{\text{DFA}} \in \text{NPSPACE}$

Theorem: $LADDER_{\text{DFA}} \in \text{NPSPACE}$

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.

$\le t$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER_{\text{DFA}} \in$ NPSPACE

Theorem: $LADDER_{\text{DFA}} \in$ 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 $y \notin L(B)$.

$\leq t$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER$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 $y = v$.

$\leq t$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER$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].

$\leq t$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER$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].

Space used is for storing y and t.

$\leq t$

WORK
PORK
PORT
SORT
SOOT
SLOT
PLOT
PLOY
PLAY

$LADDER_{\text{DFA}} \in$ NPSPACE

Theorem: $LADDER_{\text{DFA}} \in$ NPSPACE

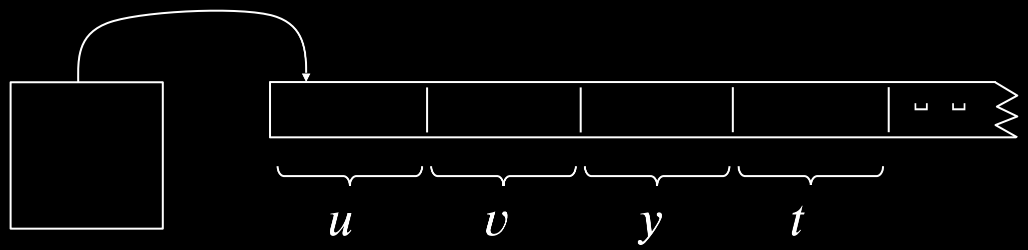
Proof idea: Nondeterministically guess the sequence from u to v.

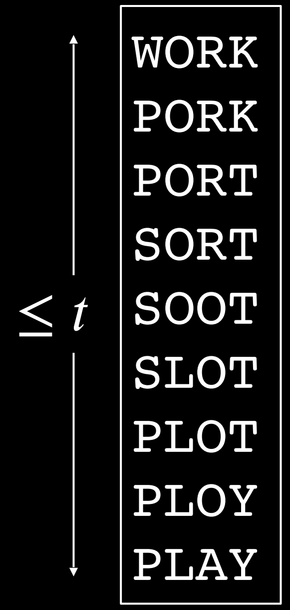
Careful- (a) cannot store sequence, (b) must terminate.

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].

Space used is for storing y and t.





$LADDER$DFA $\in$ NPSPACE

Theorem: $LADDER$DFA $\in$ NPSPACE

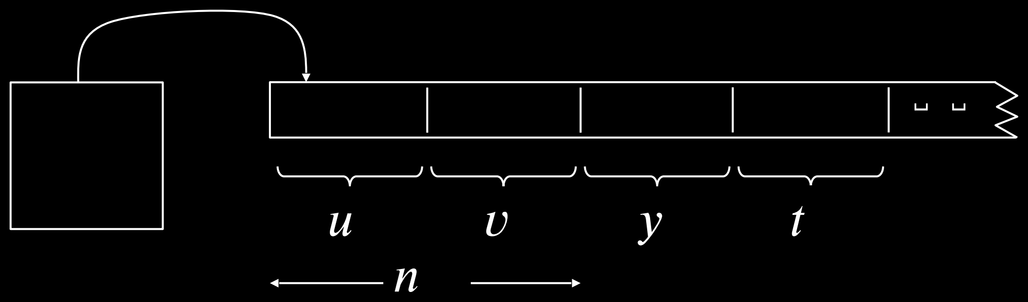
Proof idea: Nondeterministically guess the sequence from u to v.

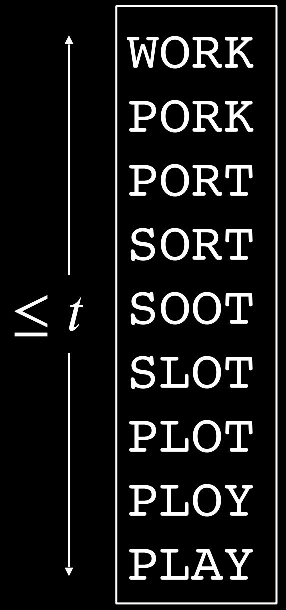
Careful- (a) cannot store sequence, (b) must terminate.

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].

Space used is for storing y and t.





$LADDER$DFA $\in$ NPSPACE

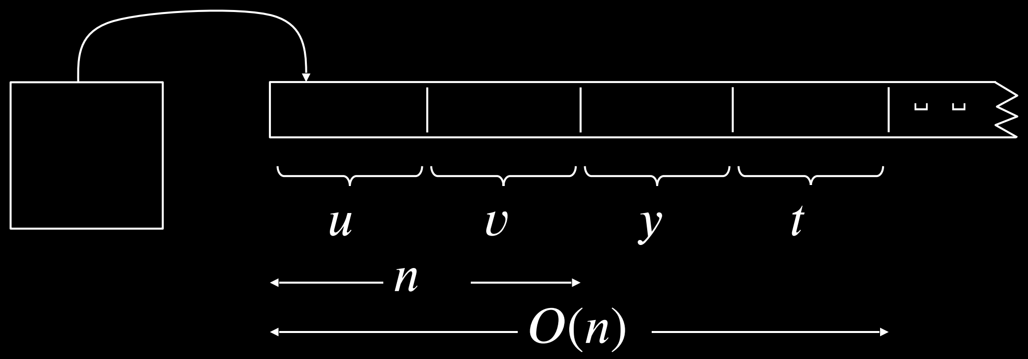
Theorem: $LADDER$DFA $\in$ NPSPACE

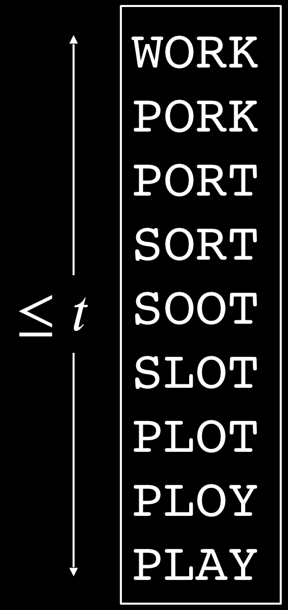
Proof idea: Nondeterministically guess the sequence from u to v.

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].





Space used is for storing y and t.

$LADDER$DFA $\in$ NPSPACE

Theorem: $LADDER$DFA $\in$ NPSPACE

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

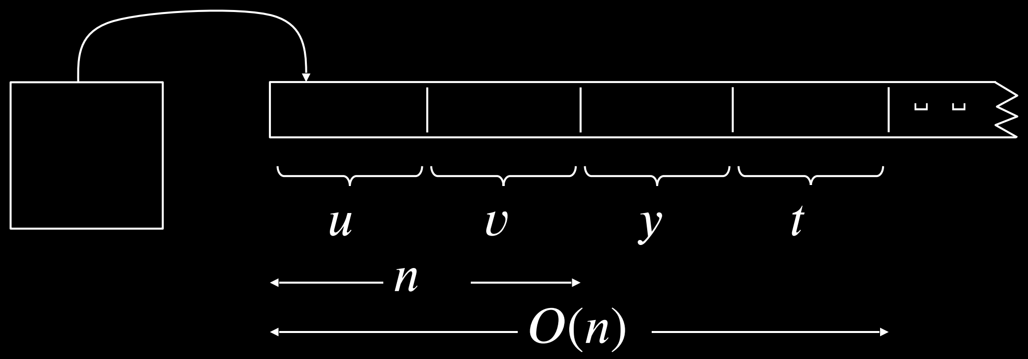
Careful- (a) cannot store sequence, (b) must terminate.

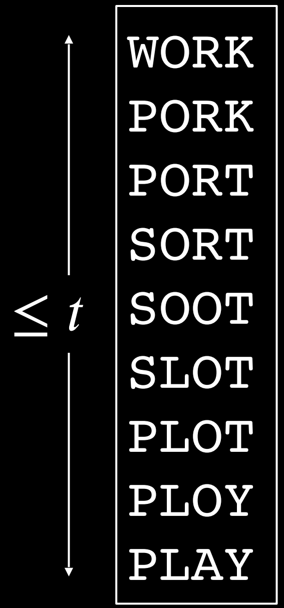
Proof: "On input $\langle B, u, v \rangle$

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].

Space used is for storing y and t.

$LADDER$DFA $\in$ NSPACE(n).





$LADDER$DFA $\in$ NPSPACE

Theorem: $LADDER$DFA $\in$ NPSPACE

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

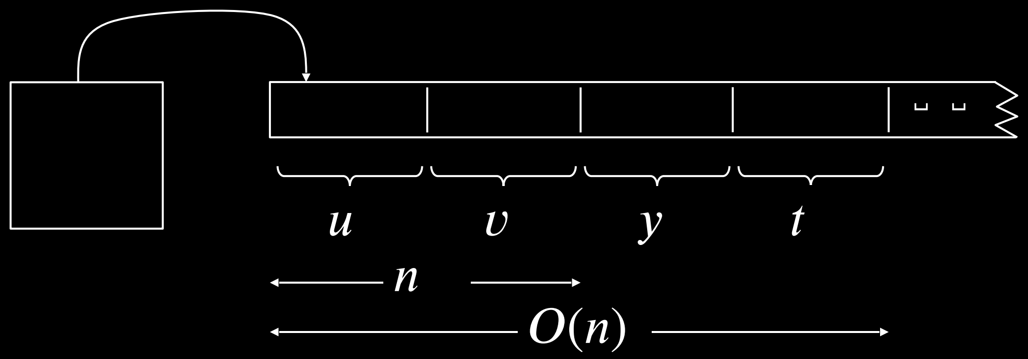
Careful- (a) cannot store sequence, (b) must terminate.

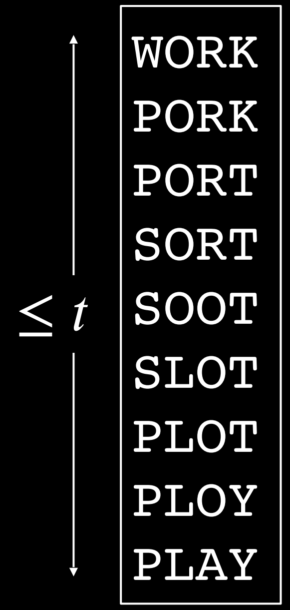
Proof: "On input $\langle B, u, v\rangle$

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].

Space used is for storing y and t.

$LADDER$DFA $\in$ NSPACE(n).





Theorem: $LADDER$DFA $\in$ PSPACE (!)

$LADDER$DFA $\in$ NPSPACE

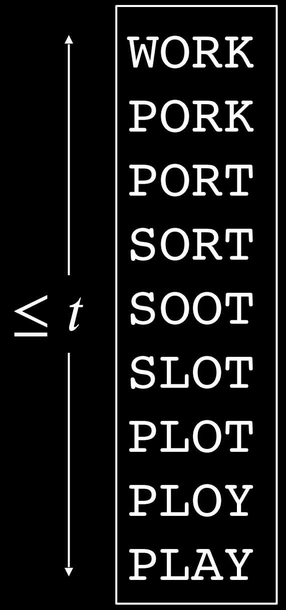
Theorem: $LADDER$DFA $\in$ NPSPACE

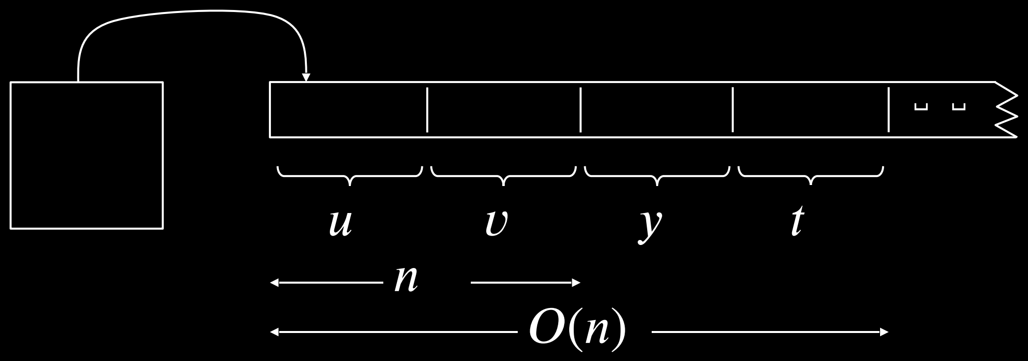
Proof idea: Nondeterministically guess the sequence from u to v.

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].





Space used is for storing y and t.

$LADDER$DFA $\in$ NSPACE(n).

Theorem: $LADDER$DFA $\in$ PSPACE (!)

$\mathit{LADDER}_{\mathrm{DFA}} \in \mathrm{PSPACE}$

$LADDER_{\text{DFA}} \in$ PSPACE

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

$LADDER$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 $\leq b$.

$LADDER_{\text{DFA}} \in \text{PSPACE}$

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

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:

$LADDER_{\text{DFA}} \in$ PSPACE

Theorem: $LADDER_{\text{DFA}} \in$ 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:

$BOUNDED\text{-}LADDER_{\text{DFA}} = \left\{ \langle B, u, v, b \rangle \,\middle|\, B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \right\}$

$LADDER_{\text{DFA}} \in$ PSPACE

Theorem: $LADDER_{\text{DFA}} \in$ 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:

$BOUNDED\text{-}LADDER_{\text{DFA}} = \left\{ \langle B, u, v, b \rangle \,\middle|\, B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \right\}$

WORK

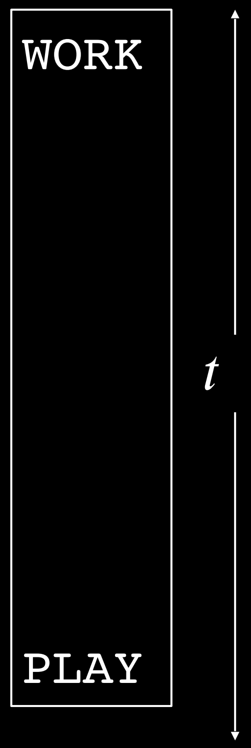
PLAY

$LADDER$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:

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

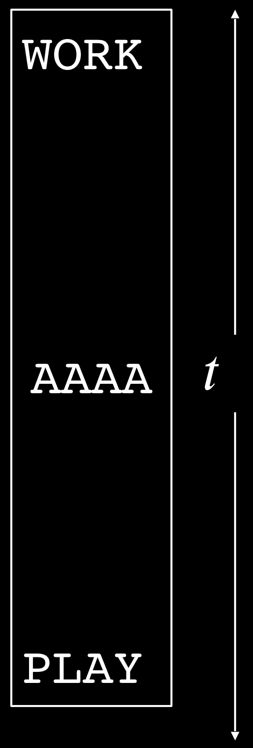


$LADDER$DFA $\in$ PSPACE

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:

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

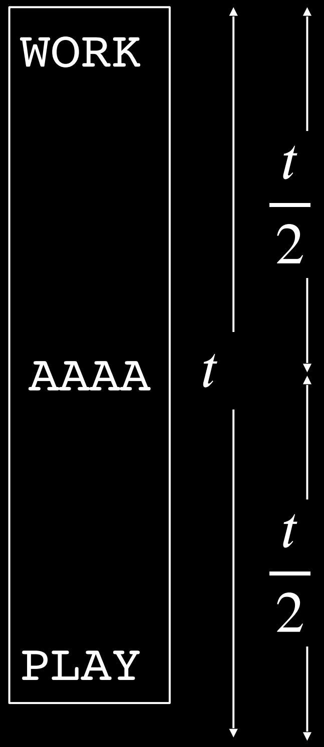


$LADDER$DFA $\in$ PSPACE

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:

$BOUNDED\text{-}LADDER$DFA $= \left\{ \langle B, u, v, b \rangle \middle| \; B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \right\}$

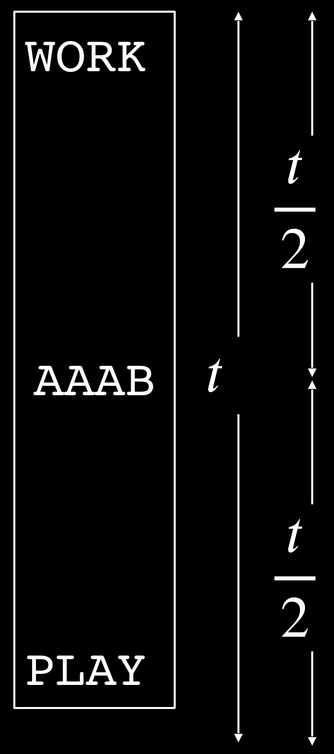


$LADDER$DFA $\in$ PSPACE

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:

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

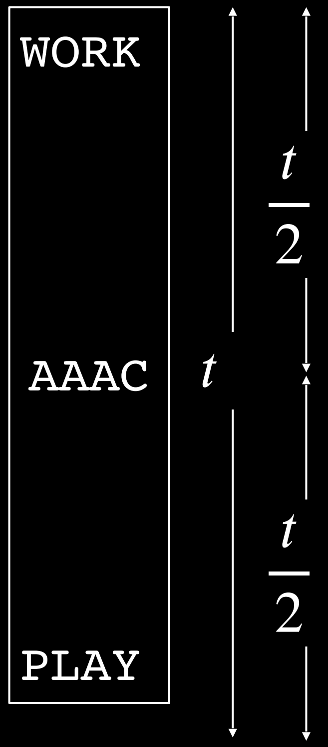


$LADDER$DFA $\in$ PSPACE

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:

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

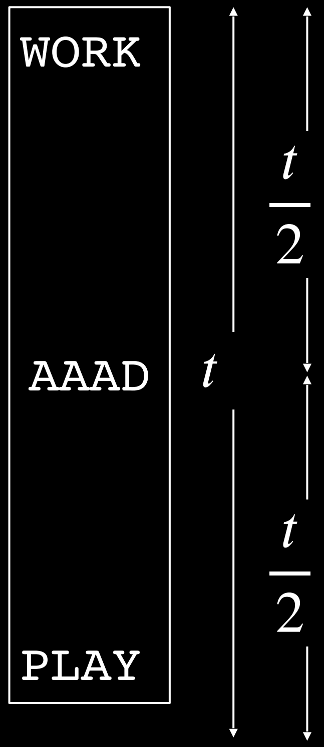


$LADDER_{\text{DFA}} \in$ PSPACE

Theorem: $LADDER_{\text{DFA}} \in$ 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:

$$BOUNDED\text{-}LADDER_{\text{DFA}} = \left\{\langle B, u, v, b\rangle \,\middle|\, B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B)\right\}$$

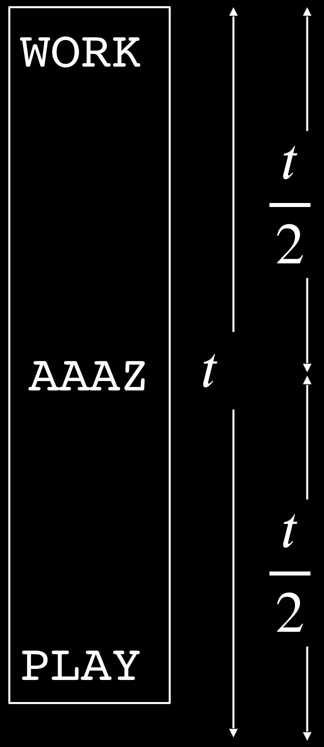


$LADDER$DFA $\in$ PSPACE

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:

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

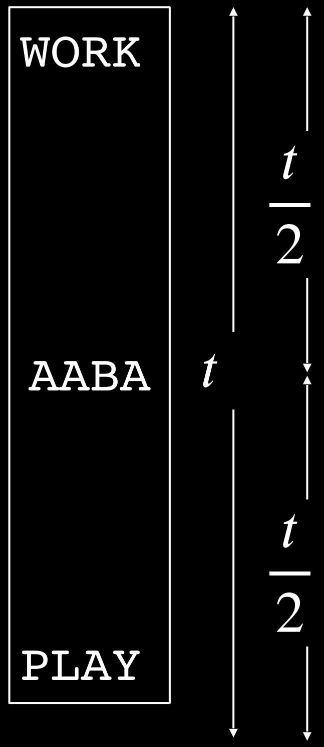


$LADDER$DFA $\in$ PSPACE

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:

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

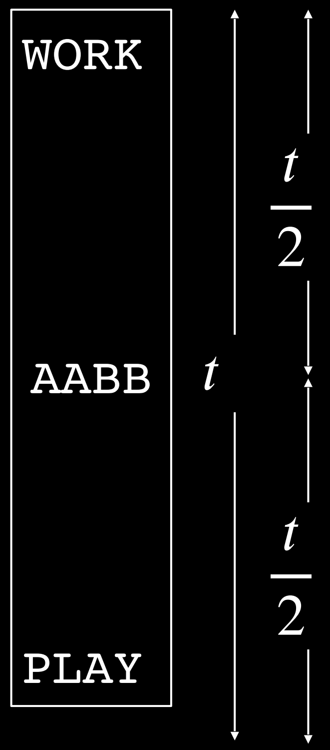


$LADDER$DFA $\in$ PSPACE

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:

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

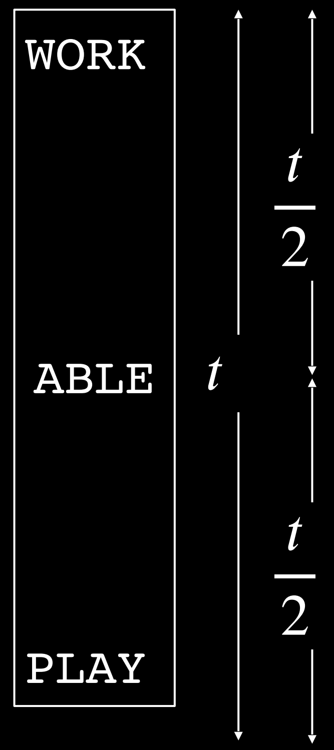


$LADDER$DFA $\in$ PSPACE

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:

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

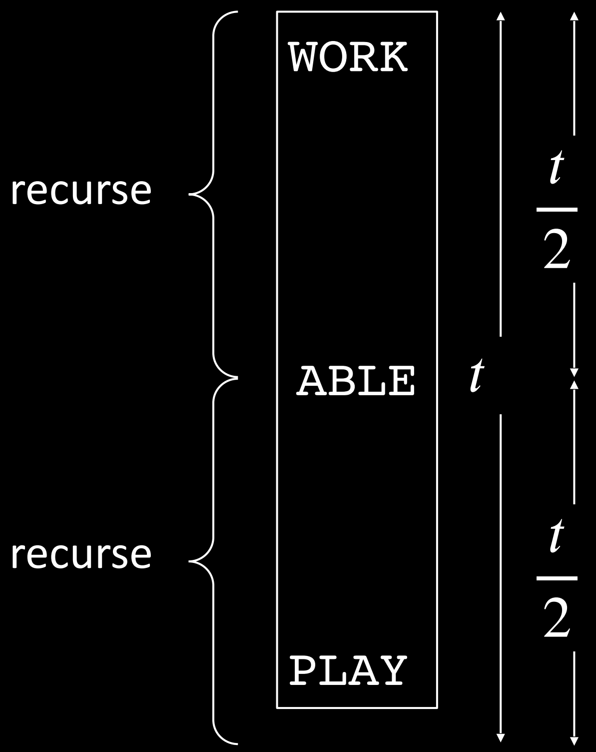


$LADDER$DFA $\in$ PSPACE

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:

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



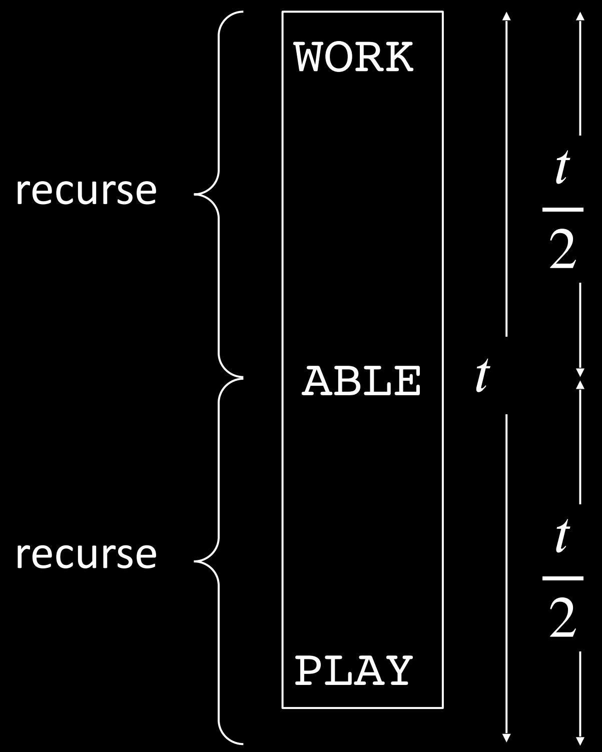
$LADDER$DFA $\in$ PSPACE

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:

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

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



$LADDER$DFA $\in$ PSPACE

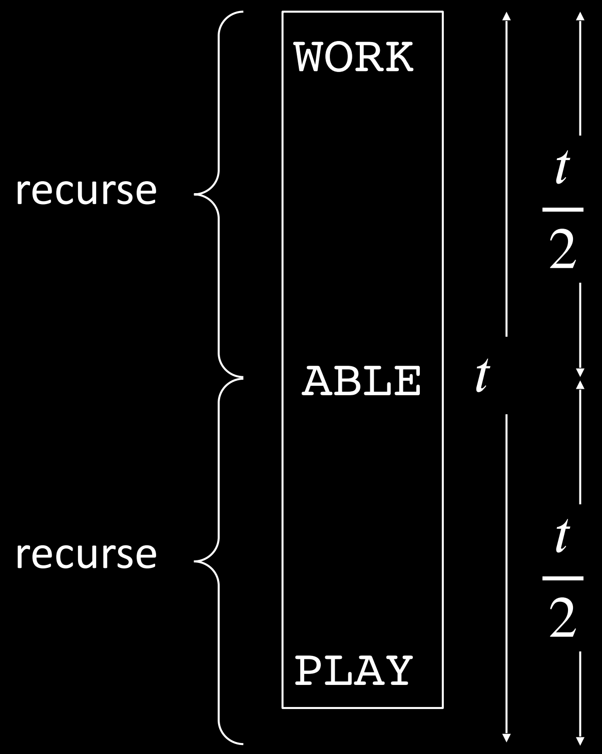
Theorem: $LADDER$DFA $\in$ SPACE(n^2)

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:

$BOUNDED\text{-}LADDER$DFA $= \{\langle B, u, v, b\rangle \mid B$ a DFA and $u \xrightarrow{b} v$ 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*.



$LADDER$DFA $\in$ PSPACE

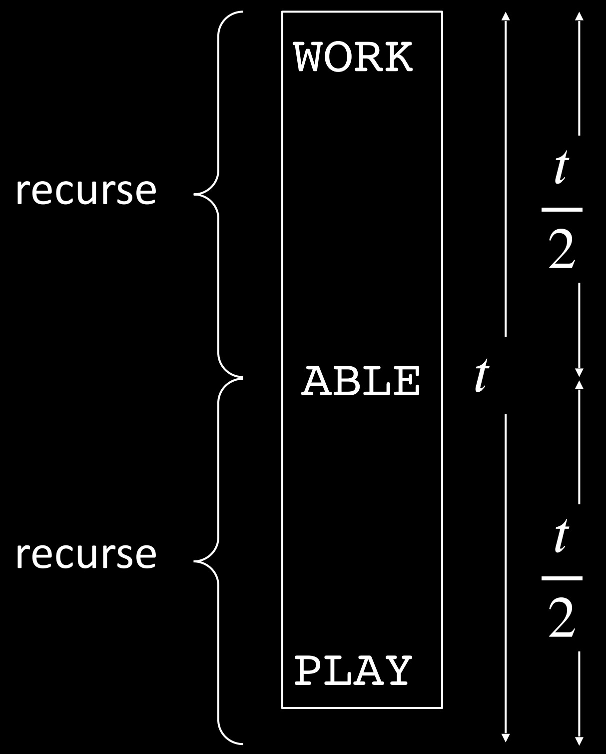
Theorem: $LADDER$DFA $\in$ SPACE(n^2)

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:

$BOUNDED\text{-}LADDER$DFA $= \{\langle B, u, v, b\rangle \mid B$ a DFA and $u \xrightarrow{b} v$ 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 $|u|$



$LADDER_{\text{DFA}} \in$ PSPACE

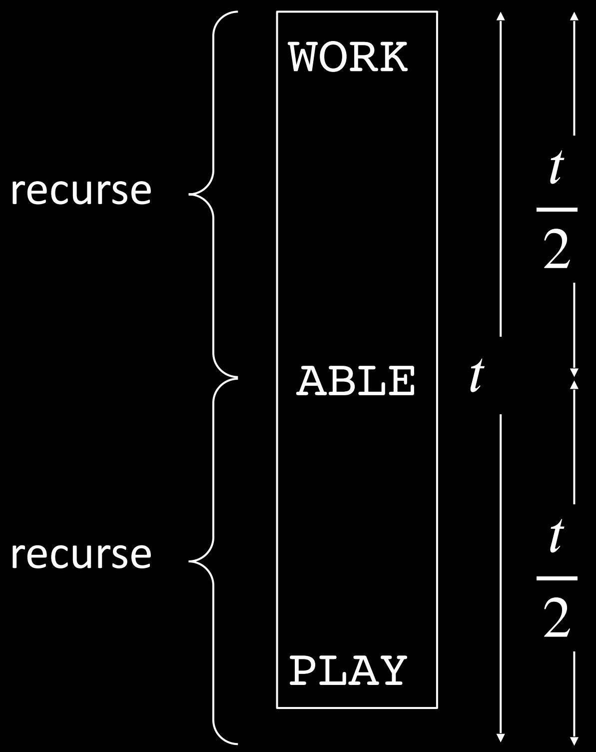
Theorem: $LADDER_{\text{DFA}} \in$ SPACE(n^2)

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:

$BOUNDED\text{-}LADDER_{\text{DFA}} = \left\{\langle B, u, v, b\rangle \,\middle|\, B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B)\right\}$

$B\text{-}L =$ "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]



$LADDER$DFA $\in$ PSPACE

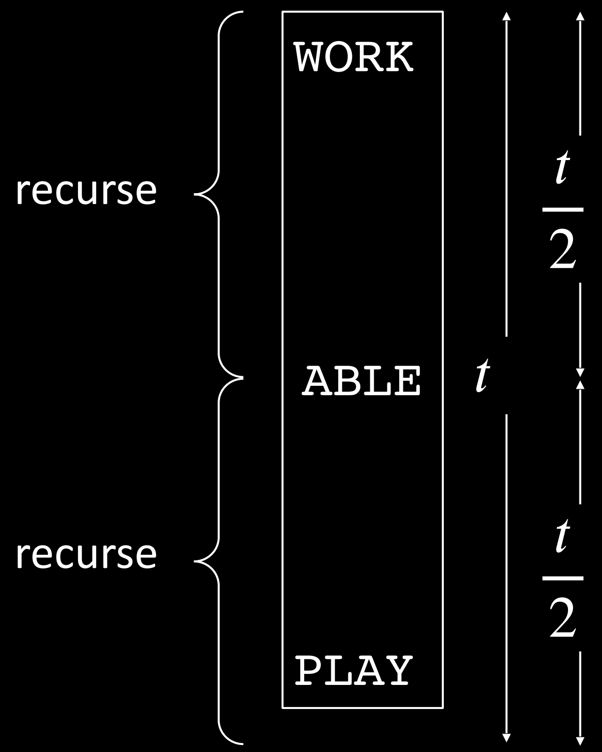
Theorem: $LADDER$DFA $\in$ SPACE(n^2)

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:

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

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.



$LADDER$DFA $\in$ PSPACE

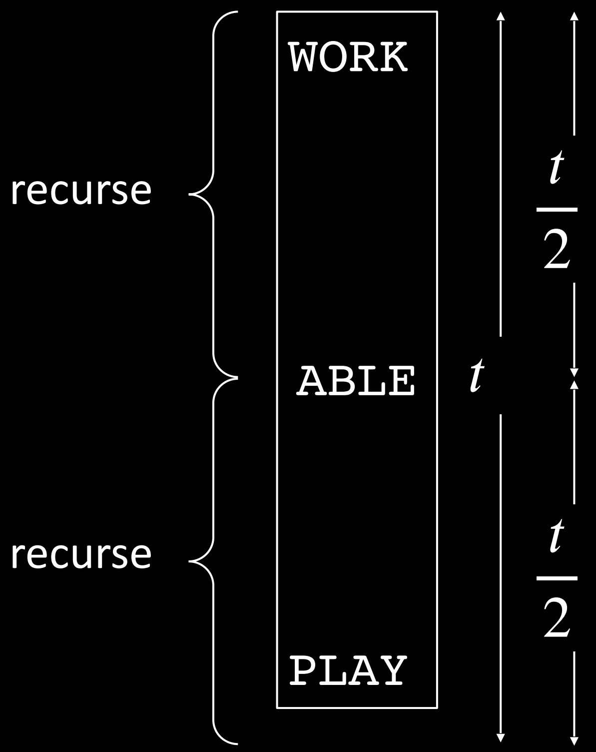
Theorem: $LADDER$DFA $\in$ SPACE(n^2)

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:

$BOUNDED\text{-}LADDER$DFA $= \{\langle B, u, v, b\rangle \mid B$ a DFA and $u \xrightarrow{b} v$ 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]."



$LADDER$DFA $\in$ PSPACE

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

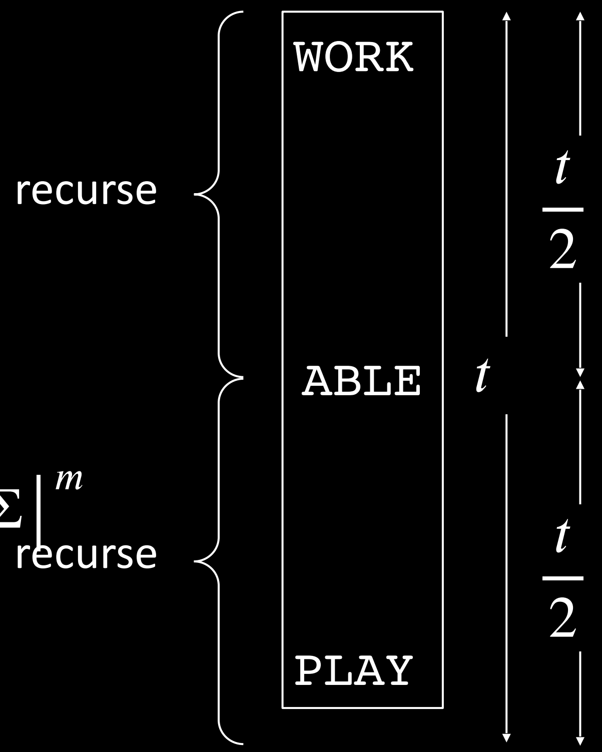
Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.
Here's a recursive procedure to solve the bounded DFA ladder problem:

$BOUNDED\text{-}LADDER$DFA $= \{\langle B, u, v, b \rangle \mid B$ a DFA and $u \xrightarrow{b} v$ 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 $\langle B, u, v \rangle \in LADDER$DFA with $B\text{-}L$ procedure on input $\langle B, u, v, t \rangle$ for $t = |\Sigma|^m$



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:

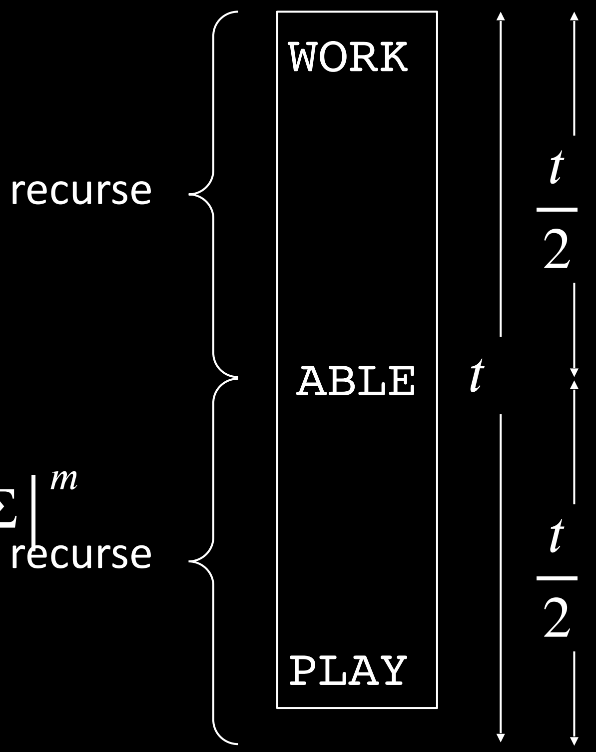
$\mathit{BOUNDED\text{-}LADDER}$DFA $= \left\{ \langle B, u, v, b \rangle \,\middle|\, B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \right\}$

$B\text{-}L =$ "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 $\langle B, u, v \rangle \in \mathit{LADDER}$DFA with $B\text{-}L$ procedure on input $\langle B, u, v, t \rangle$ for $t = |\Sigma|^m$

Space analysis:



$LADDER_{\text{DFA}} \in$ PSPACE

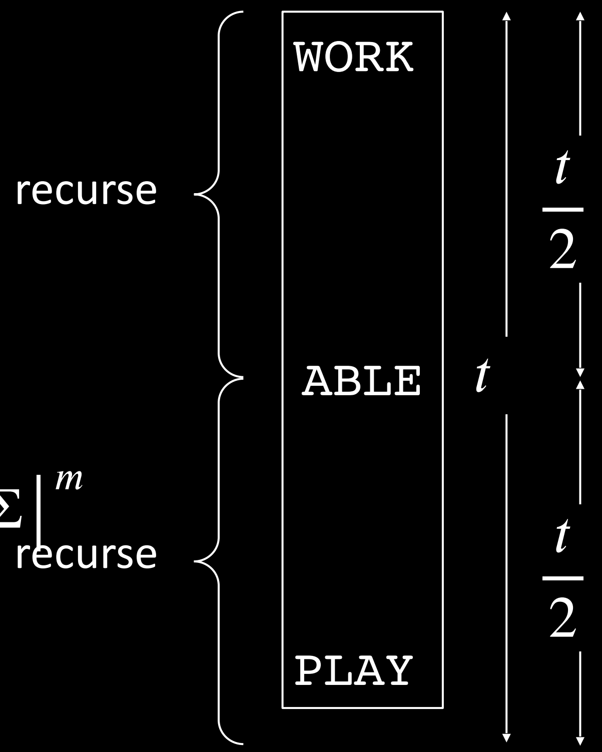
Theorem: $LADDER_{\text{DFA}} \in$ SPACE(n^2)

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:

$BOUNDED\text{-}LADDER_{\text{DFA}} = \left\{\langle B, u, v, b\rangle \,\middle|\, B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B)\right\}$

$B\text{-}L =$ "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 $\langle B, u, v\rangle \in LADDER_{\text{DFA}}$ with $B\text{-}L$ procedure on input $\langle B, u, v, t\rangle$ for $t = |\Sigma|^m$

Space analysis:

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

$LADDER$DFA $\in$ PSPACE

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

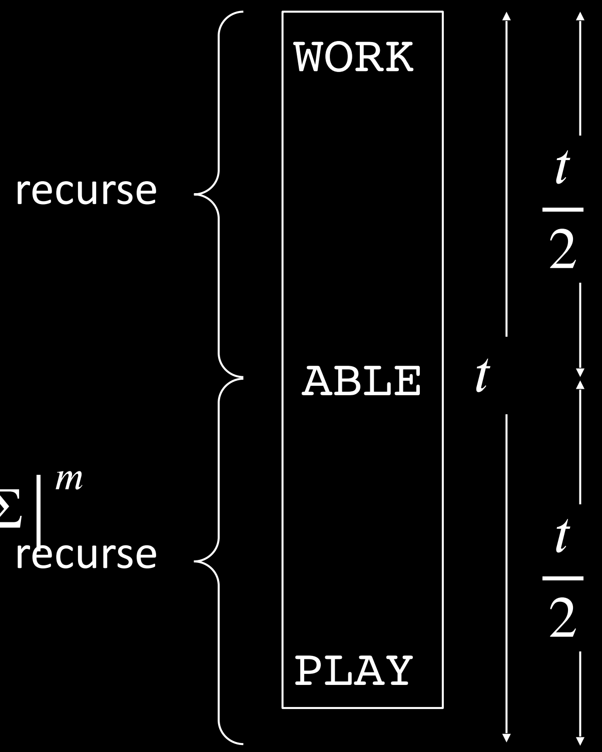
Proof: Write $u \xrightarrow{b} v$ if there's a ladder from u to v of length $\leq b$.
Here's a recursive procedure to solve the bounded DFA ladder problem:

$BOUNDED\text{-}LADDER$DFA $= \left\{ \langle B, u, v, b \rangle \middle| \; B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \right\}$

$B\text{-}L =$ "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 $\langle B, u, v \rangle \in LADDER$DFA with $B\text{-}L$ procedure on input $\langle B, u, v, t \rangle$ for $t = |\Sigma|^m$



Space analysis:

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

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

$LADDER$DFA $\in$ PSPACE

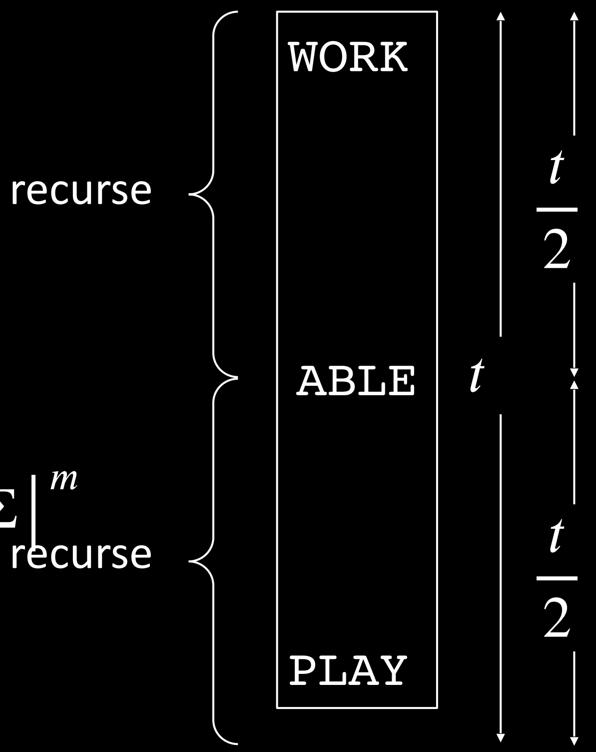
Theorem: $LADDER$DFA $\in$ SPACE(n^2)

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:

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

$B\text{-}L =$ "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 $\langle B, u, v\rangle \in LADDER$DFA with $B\text{-}L$ procedure on input $\langle B, u, v, t\rangle$ for $t = |\Sigma|^m$

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 $O(n^2)$.

$LADDER_{\text{DFA}} \in \text{PSPACE}$

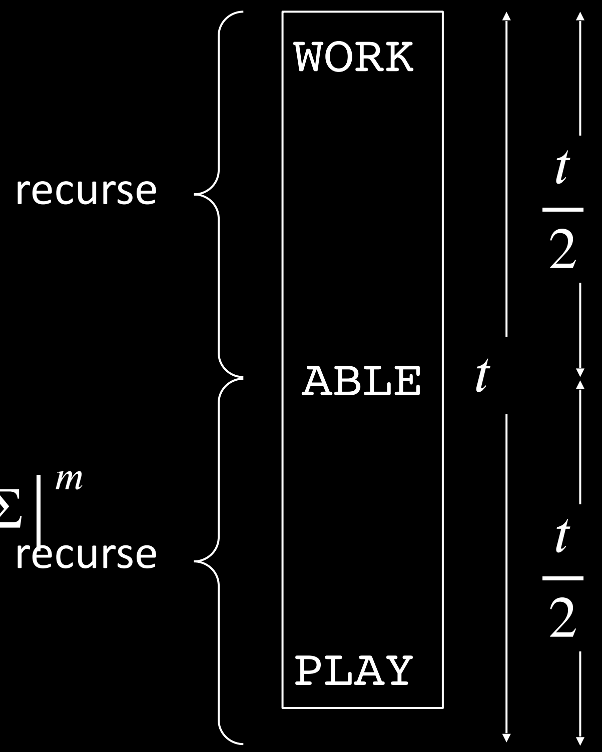
Theorem: $LADDER_{\text{DFA}} \in \text{SPACE}(n^2)$

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:

$BOUNDED\text{-}LADDER_{\text{DFA}} = \left\{ \langle B, u, v, b \rangle \,\middle|\, B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B) \right\}$

$B\text{-}L =$ "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 $\langle B, u, v \rangle \in LADDER_{\text{DFA}}$ with $B\text{-}L$ procedure on input $\langle B, u, v, t \rangle$ for $t = |\Sigma|^m$

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

$\mathit{LADDER}_{\text{DFA}} \in$ PSPACE

Theorem: $\mathit{LADDER}_{\text{DFA}} \in$ 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:

$$\mathit{BOUNDED\text{-}LADDER}_{\text{DFA}} = \left\{\langle B, u, v, b\rangle \middle|\ B \text{ a DFA and } u \xrightarrow{b} v \text{ by a ladder in } L(B)\right\}$$

$B\text{-}L$ = "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 $\langle B, u, v\rangle \in \mathit{LADDER}_{\text{DFA}}$ with $B\text{-}L$ 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.

Check-in 17.3

PSPACE = NPSPACE

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

PSPACE = NPSPACE

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

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

PSPACE = NPSPACE

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

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 $f(n) \geq n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

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 $f(n) \geq n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

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

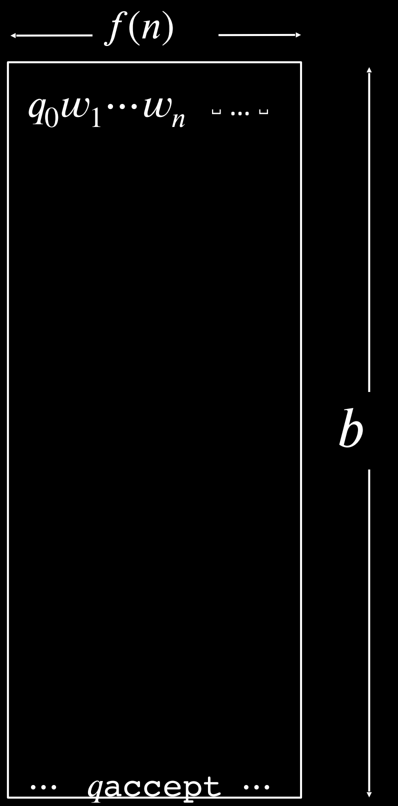
Savitch's Theorem: For $f(n) \geq n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

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 $f(n) \geq n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

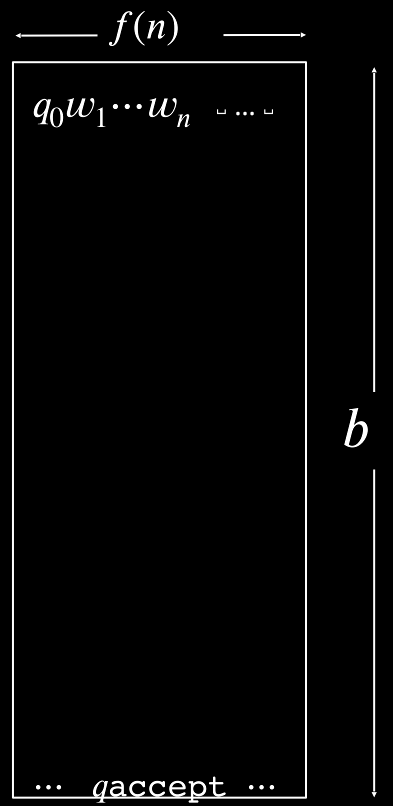
Proof: Convert NTM N to equivalent TM M, only squaring the space used.

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 $f(n) \geq n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

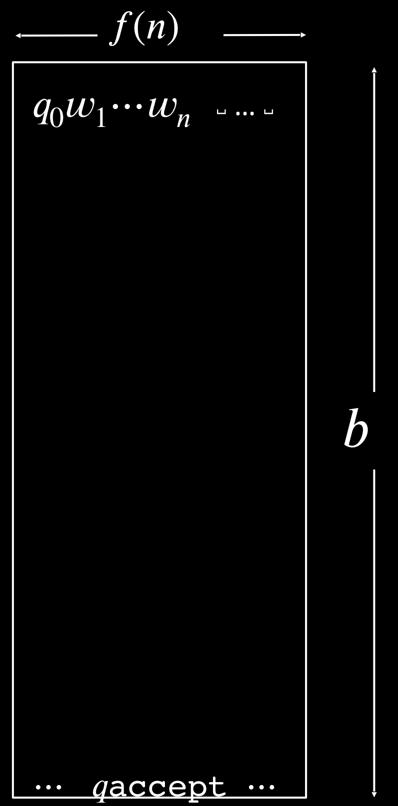
Proof: Convert NTM N to equivalent TM M, only squaring the space used.

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 $f(n) \geq n$, $\text{NSPACE}\big(f(n)\big) \subseteq \text{SPACE}\big(f^2(n)\big)$

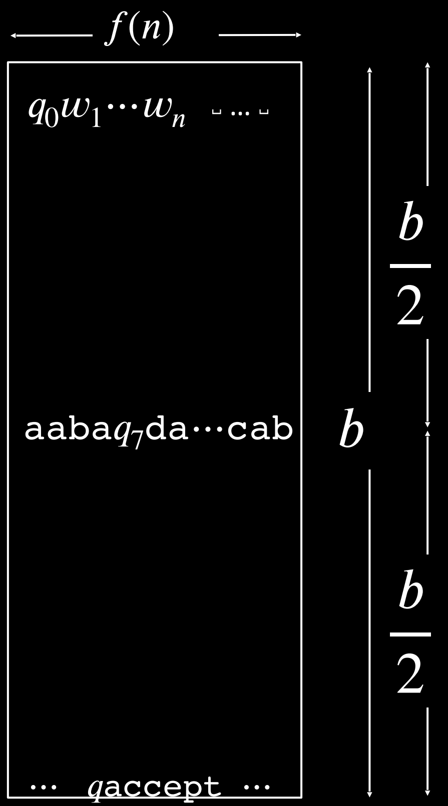
Proof: Convert NTM N to equivalent TM M, only squaring the space used.

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 $f(n) \geq n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

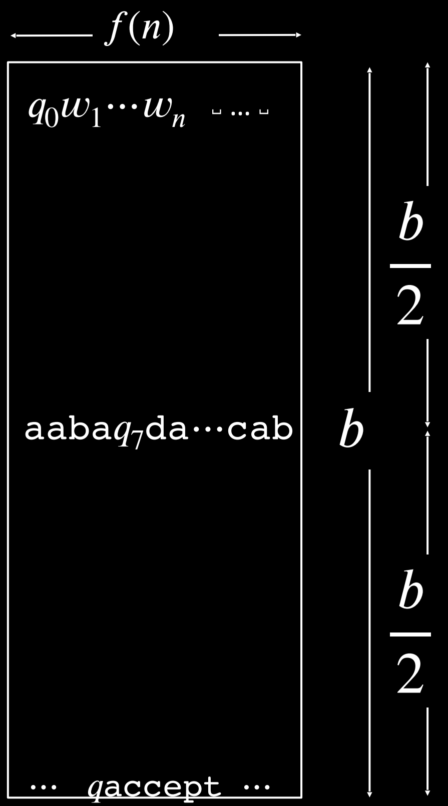
Proof: Convert NTM N to equivalent TM M, only squaring the space used.

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 $f(n) \geq n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

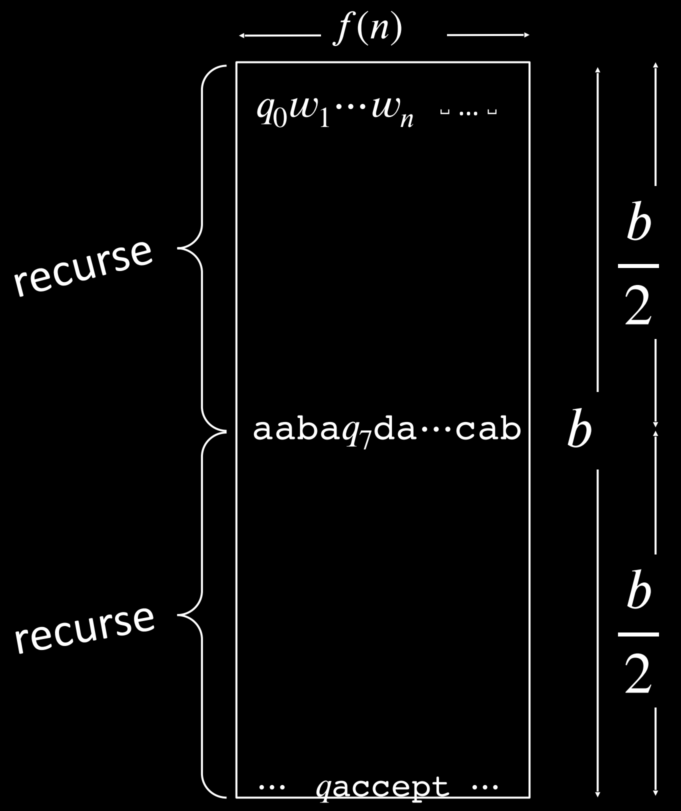
Proof: Convert NTM N to equivalent TM M, only squaring the space used.

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 $f(n) \geq n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

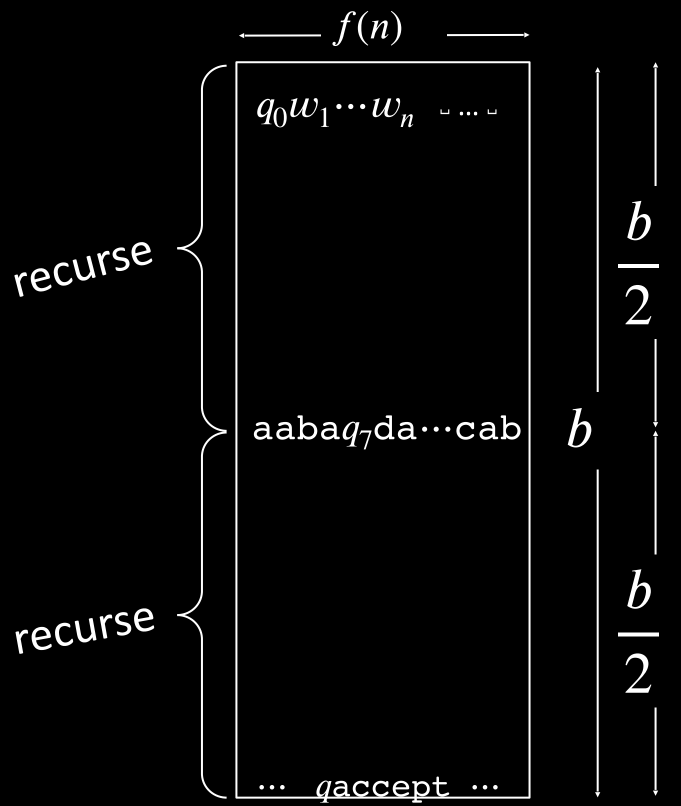
Proof: Convert NTM N to equivalent TM M, only squaring the space used.

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 $f(n) \geq n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

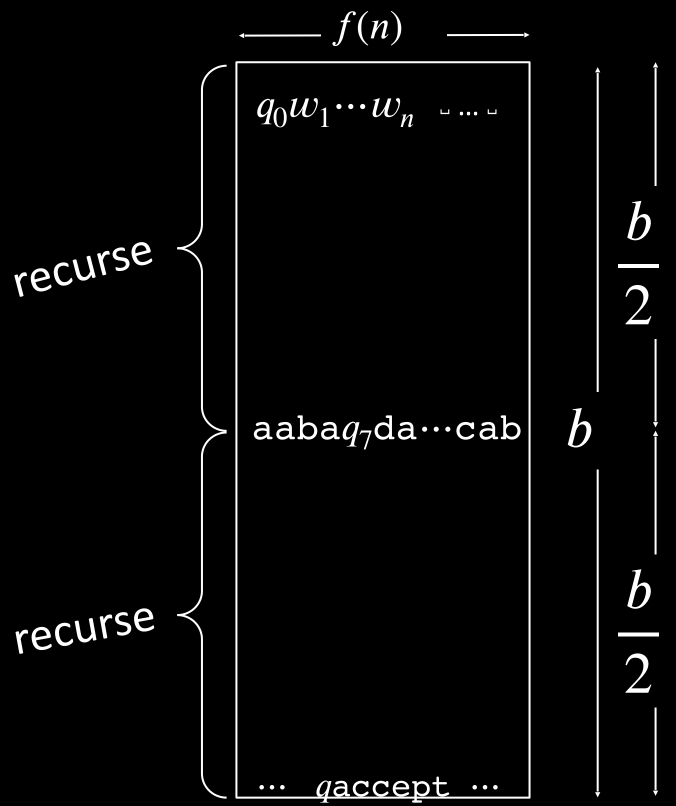
Proof: Convert NTM N to equivalent TM M, only squaring the space used.

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 $f(n) \geq n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

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

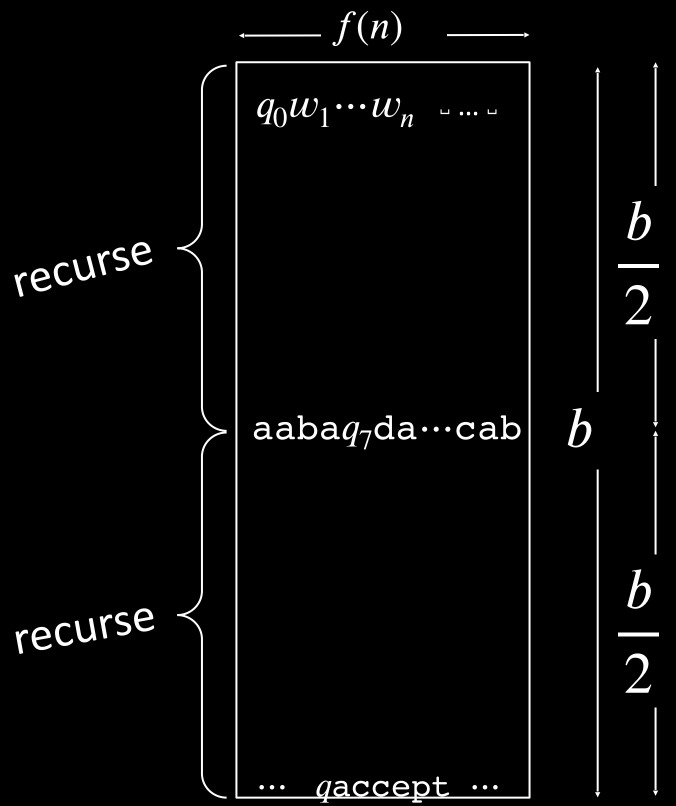
For configurations c_i and c_j of N, write $c_i \xrightarrow{b} c_j$ if can get from c_i to c_j in $\leq b$ steps.

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 $f(n) \geq n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

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.

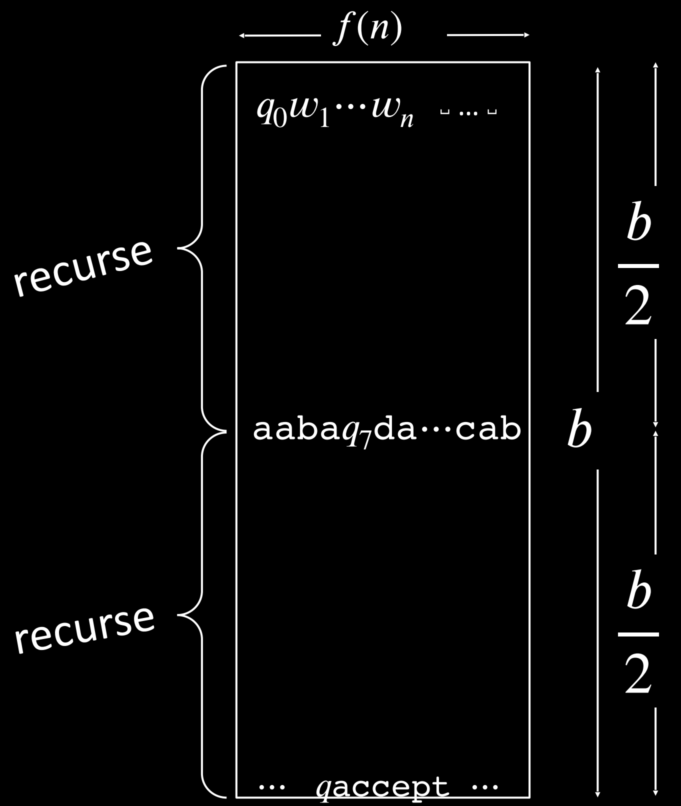
Give recursive algorithm to test $c_i \xrightarrow{b} c_j$:

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 $f(n) \geq n$, NSPACE$\left(f(n)\right) \subseteq$ SPACE$\left(f^2(n)\right)$

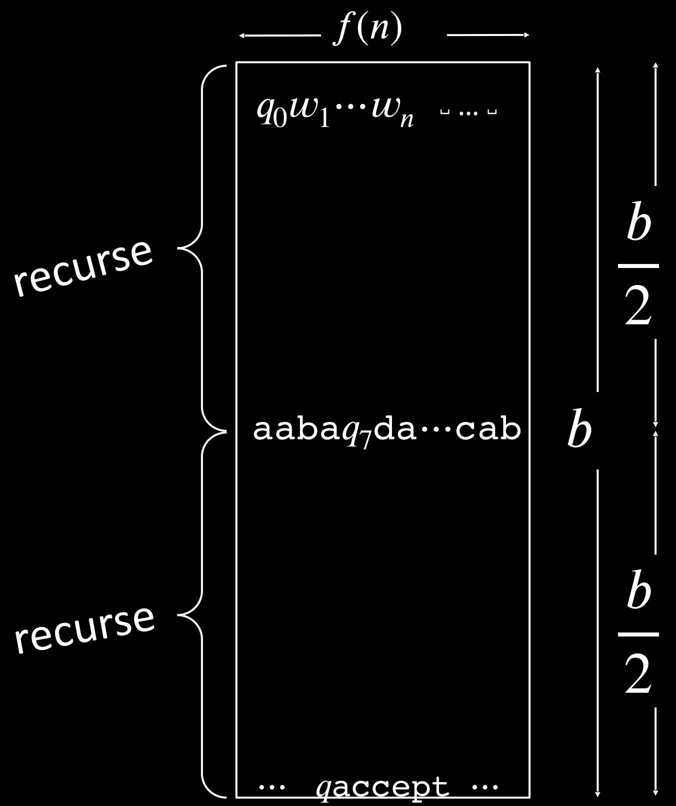
Proof: Convert NTM N to equivalent TM M, only squaring the space used.

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 = $O\left(f(n)\right)$ space.

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

PSPACE = NPSPACE

Savitch's Theorem: For $f(n) \ge n$, NSPACE$\big(f(n)\big) \subseteq$ SPACE$\big(f^2(n)\big)$

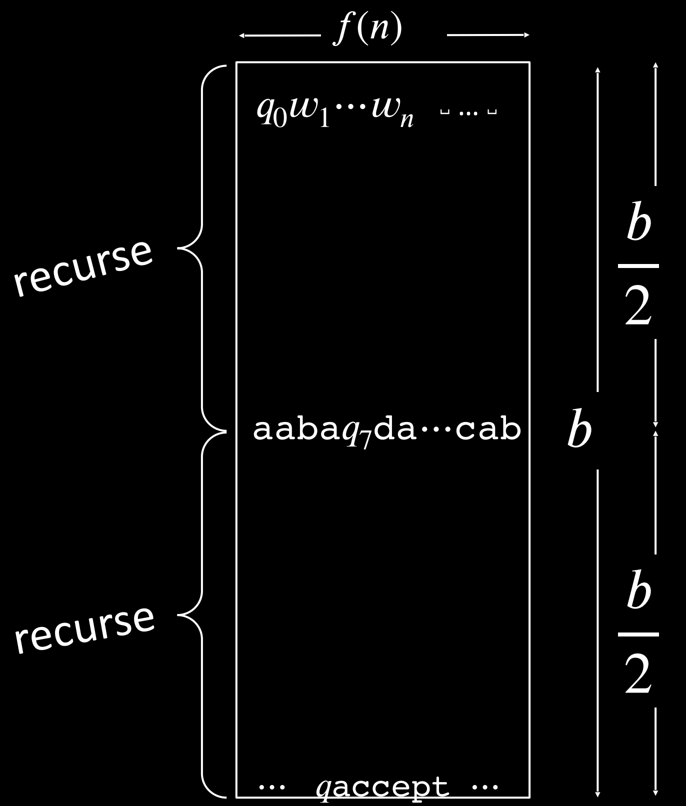
Proof: Convert NTM N to equivalent TM M, only squaring the space used.

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 $\le b$ 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 = $O\big(f(n)\big)$ space.

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

PSPACE-completeness

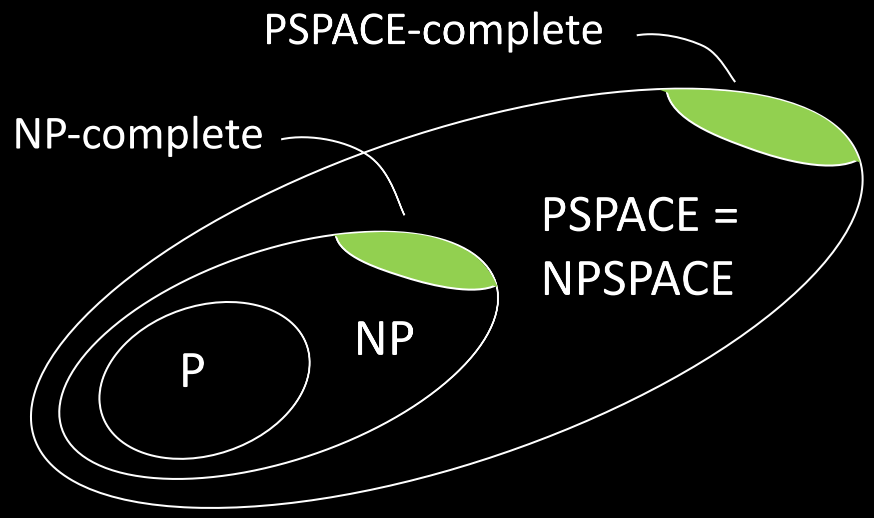
Defn: B is PSPACE-complete if

1) $B \in$ PSPACE
2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} B$

PSPACE-completeness

Defn: B is <u>PSPACE-complete</u> if

1) $B \in$ PSPACE
2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} B$



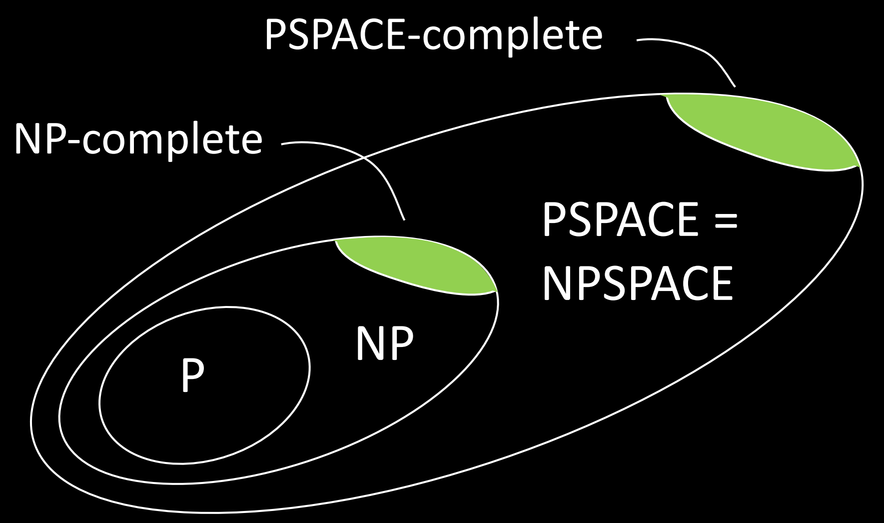
Think of complete problems as the “hardest” in their associated class.

PSPACE-completeness

Defn: B is <u>PSPACE-complete</u> if

1) $B \in$ PSPACE
2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} B$

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



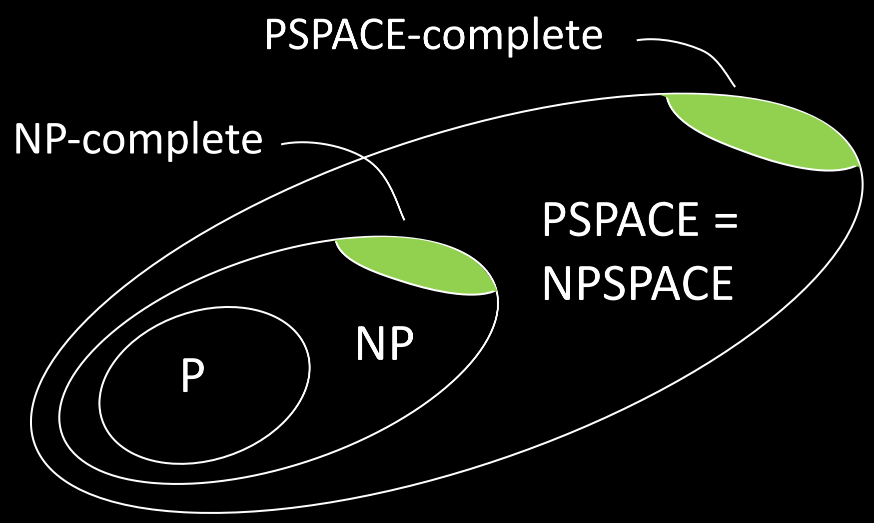
Think of complete problems as the “hardest” in their associated class.

PSPACE-completeness

Defn: B is <u>PSPACE-complete</u> if
1) $B \in$ PSPACE
2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} B$

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

Why $\leq_{\mathrm{P}}$ and not $\leq_{\mathrm{PSPACE}}$ when defining PSPACE-complete?



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

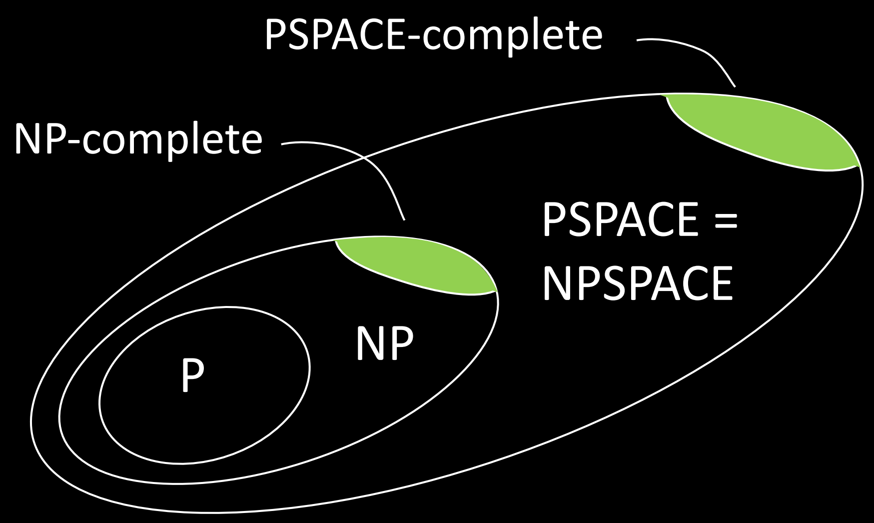
PSPACE-completeness

Defn: B is <u>PSPACE-complete</u> if

1) $B \in$ PSPACE
2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} B$

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

Why $\leq_{\mathrm{P}}$ and not $\leq_{\mathrm{PSPACE}}$ 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

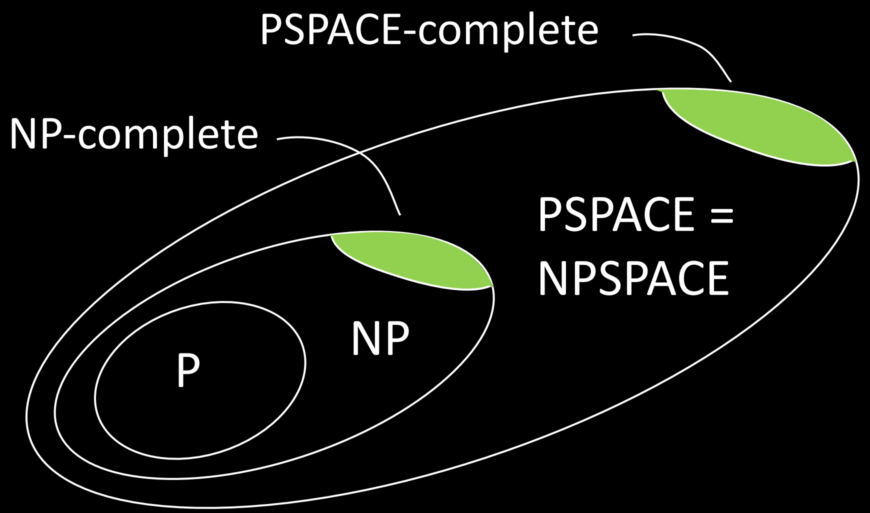
Defn: B is PSPACE-complete if

1) $B \in$ PSPACE
2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} B$

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

Why $\leq_{\mathrm{P}}$ and not $\leq_{\mathrm{PSPACE}}$ 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

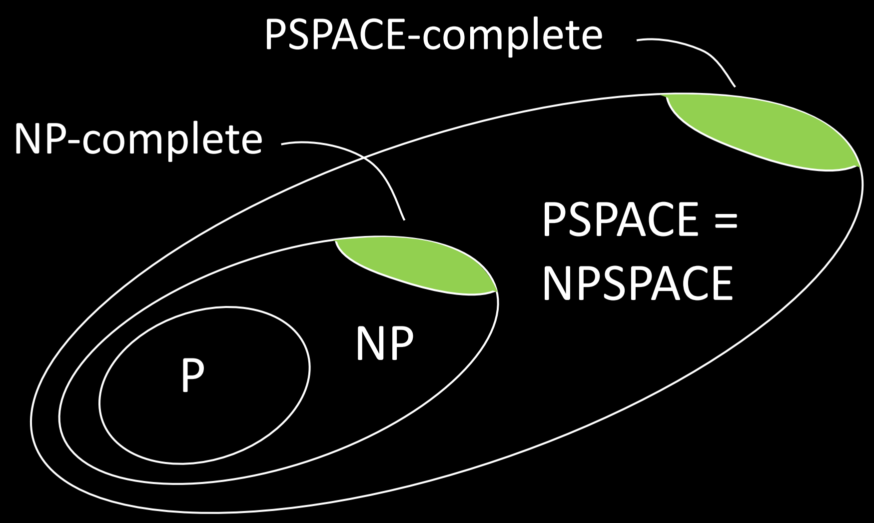
Defn: B is <u>PSPACE-complete</u> if

1) $B \in$ PSPACE
2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} B$

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

Why $\leq_{\mathrm{P}}$ and not $\leq_{\mathrm{PSPACE}}$ 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.

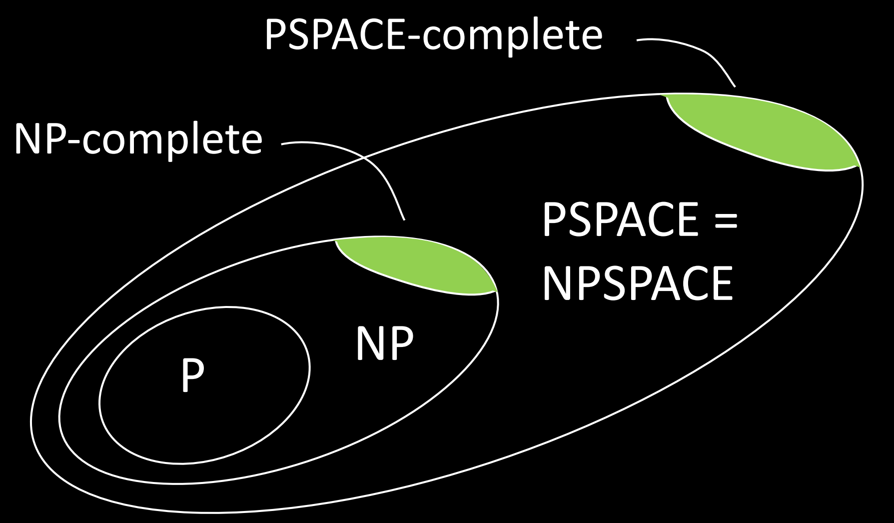
Check-in 18.1

PSPACE-completeness

Defn: B is <u>PSPACE-complete</u> if

1) $B \in$ PSPACE
2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} B$

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



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

Check-in 18.1

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

(a) P = PSPACE

(b) NP = PSPACE

(c) P = NP

(d) NP = coNP

Check-in 18.1

$TQBF$ is PSPACE-complete

Recall: $TQBF = \left\{ \langle \phi \rangle \,\middle|\, \phi \text{ is a QBF that is } \textsc{True} \right\}$

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

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

$TQBF$ is PSPACE-complete

Recall: $TQBF = \left\{ \langle \phi \rangle \,\middle|\, \phi \text{ is a QBF that is TRUE} \right\}$

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

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

Theorem: $TQBF$ is PSPACE-complete

$TQBF$ is PSPACE-complete

Recall: $TQBF = \left\{\langle\phi\rangle \,\middle|\, \phi \text{ is a QBF that is } \textsc{True}\right\}$

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

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

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in$ PSPACE ✓

$TQBF$ is PSPACE-complete

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

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

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

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in$ PSPACE ✓

2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} TQBF$

$TQBF$ is PSPACE-complete

Recall: $TQBF = \left\{\langle\phi\rangle \,\middle|\, \phi \text{ is a QBF that is } \textsc{True}\right\}$

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

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

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in$ PSPACE ✓

2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} TQBF$

Let $A \in$ 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: $TQBF = \{\langle\phi\rangle \mid \phi \text{ is a QBF that is } \textsc{True}\}$

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

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

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in$ PSPACE ✓

2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} TQBF$

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

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

$f\colon \Sigma^* \rightarrow$ QBFs

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

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

$TQBF$ is PSPACE-complete

Recall: $TQBF = \left\{ \langle \phi \rangle \middle| \ \phi \text{ is a QBF that is } \textsc{True} \right\}$

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

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

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in$ PSPACE ✓

2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} TQBF$

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

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

$f \colon \Sigma^* \to$ QBFs

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

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

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

$TQBF$ is PSPACE-complete

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

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

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

Theorem: $TQBF$ is PSPACE-complete

Proof: 1) $TQBF \in$ PSPACE ✓

2) For all $A \in$ PSPACE, $A \leq_{\mathrm{P}} TQBF$

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

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

$f\colon \Sigma^* \rightarrow$ QBFs

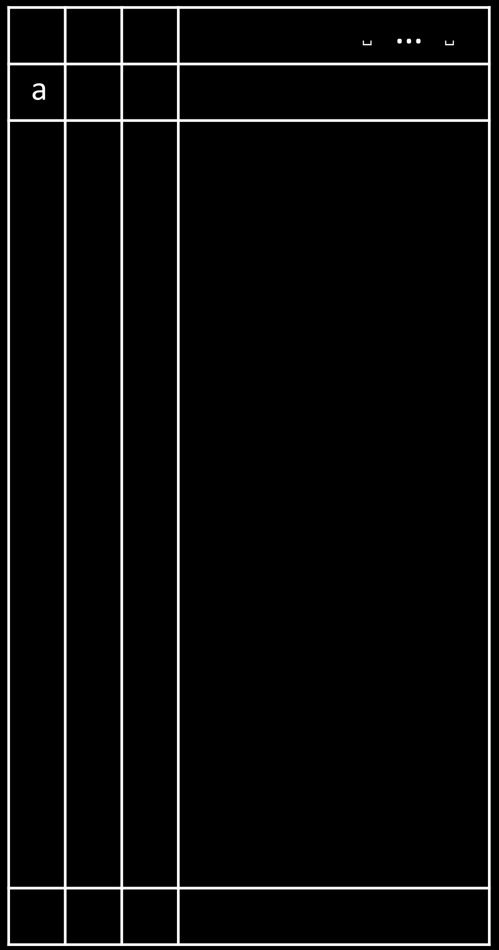
$f(w) = \langle\phi_{M,w}\rangle$

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

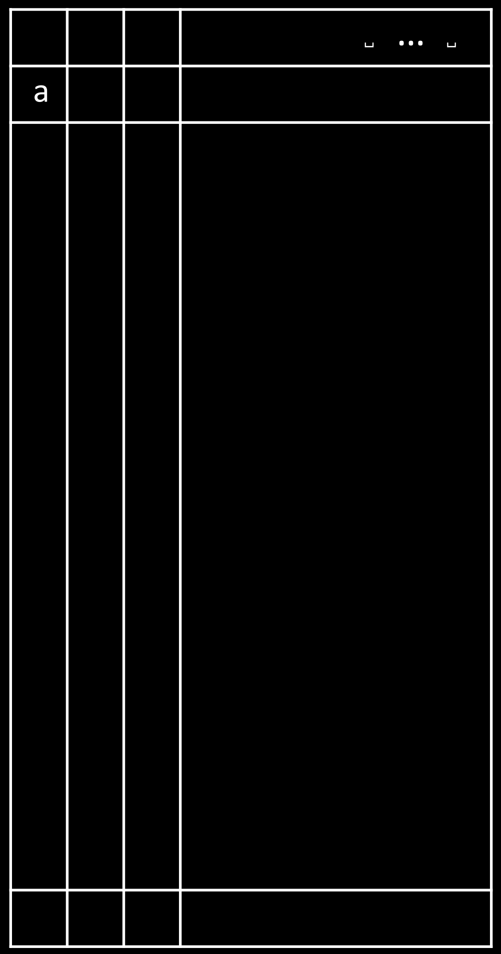


Recall: A tableau for M on w represents a computation history for M on w when M accepts w.

Rows of that tableau are configurations.

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

Tableau for M on w

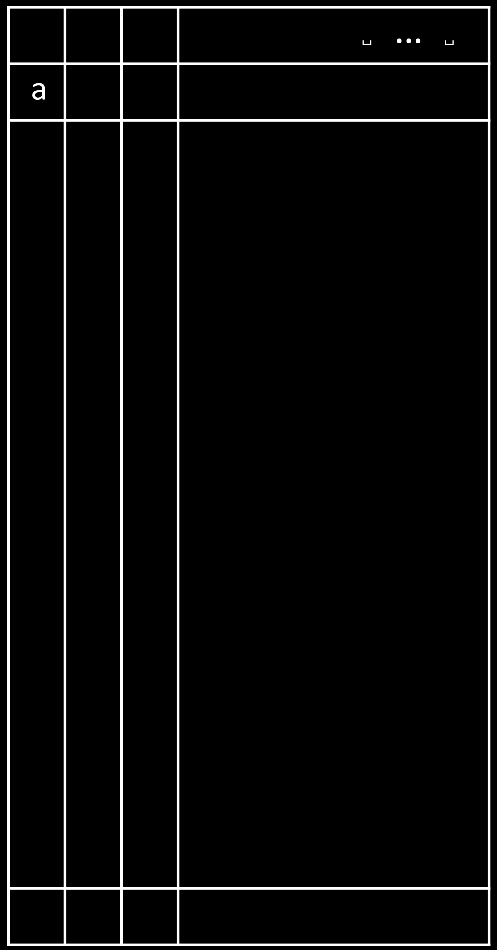


Recall: A tableau for M on w represents a computation history for M on w when M accepts w.
Rows of that tableau are configurations.

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

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

Tableau for M on w



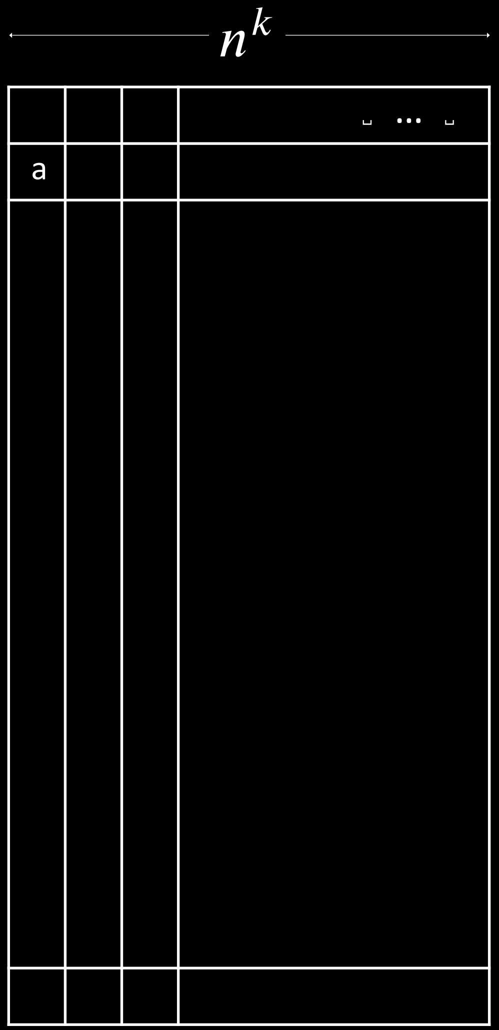
Recall: A tableau for M on w represents a computation history for M on w when M accepts w.
Rows of that tableau are configurations.

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



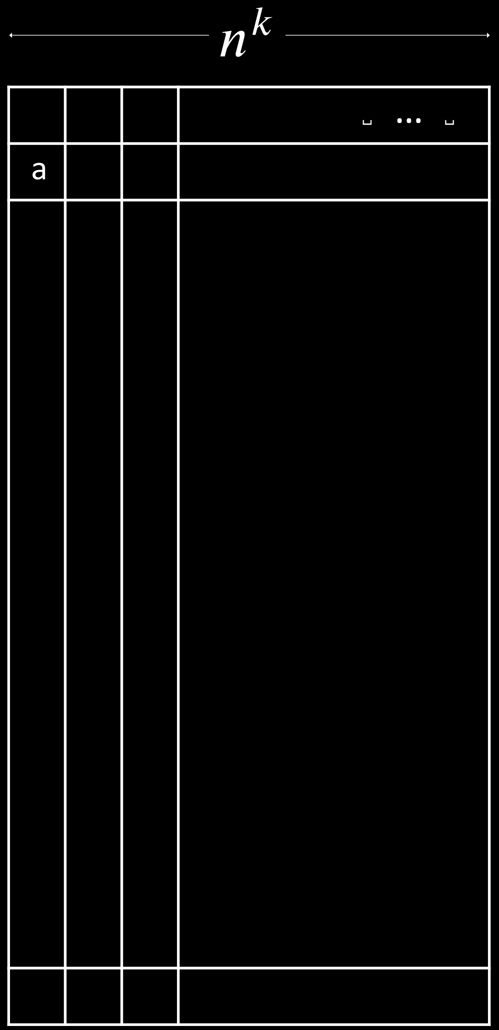
Recall: A tableau for M on w represents a computation history for M on w when M accepts w.
Rows of that tableau are configurations.

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



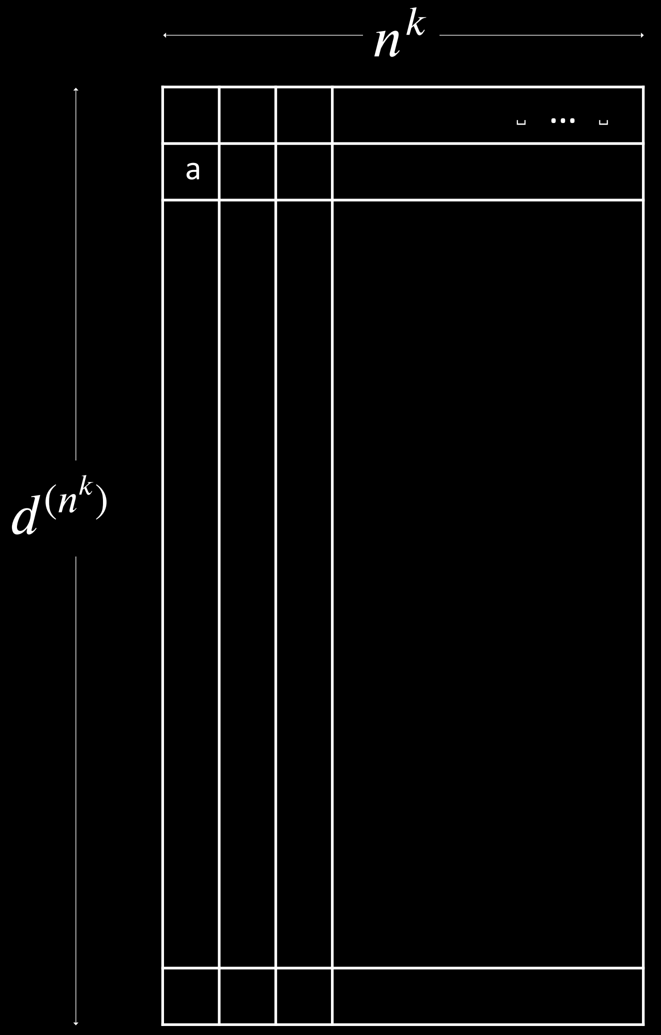
Recall: A tableau for M on w represents a computation history for M on w when M accepts w.
Rows of that tableau are configurations.

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



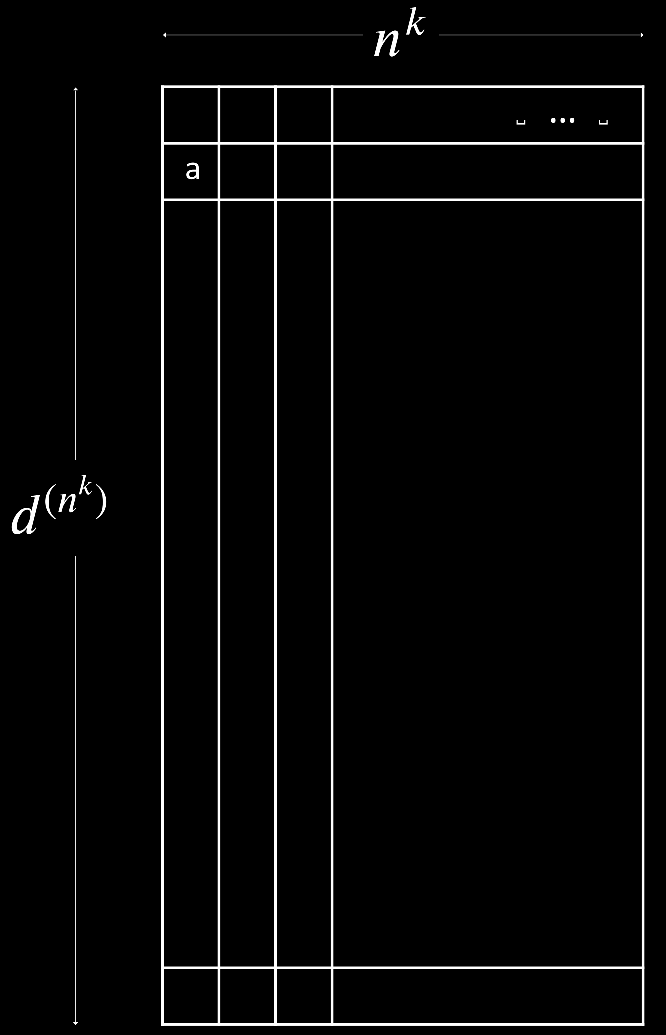
Recall: A tableau for M on w represents a computation history for M on w when M accepts w.
Rows of that tableau are configurations.

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



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.

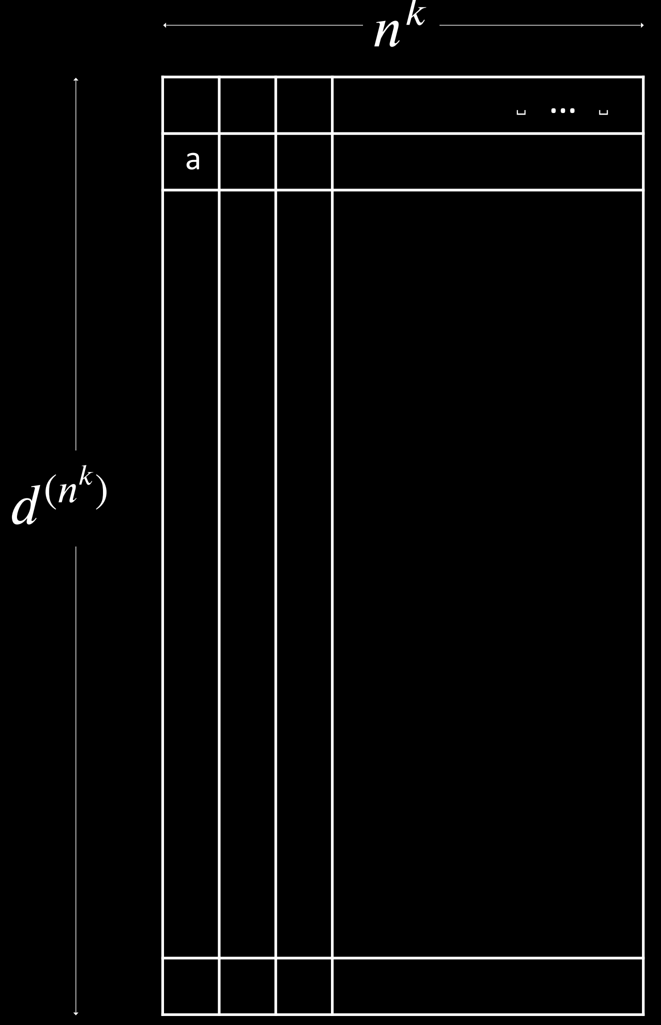
M runs in space n^k, its tableau has:

- 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



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:

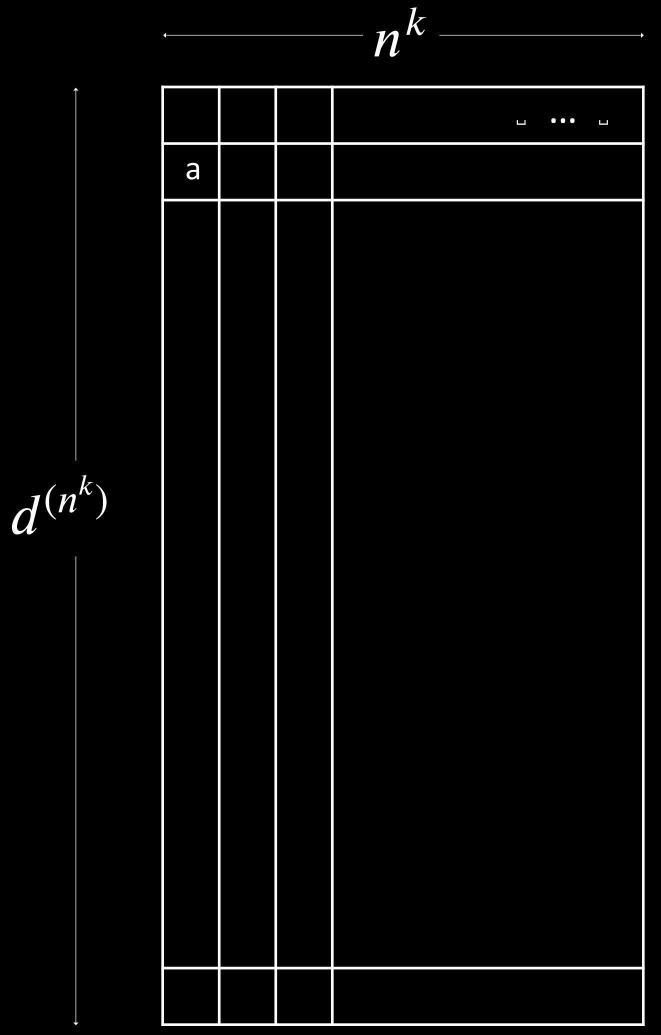
- n^k columns (max size of a configuration)
- $d^{(n^k)}$ rows (max number of steps)

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



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)

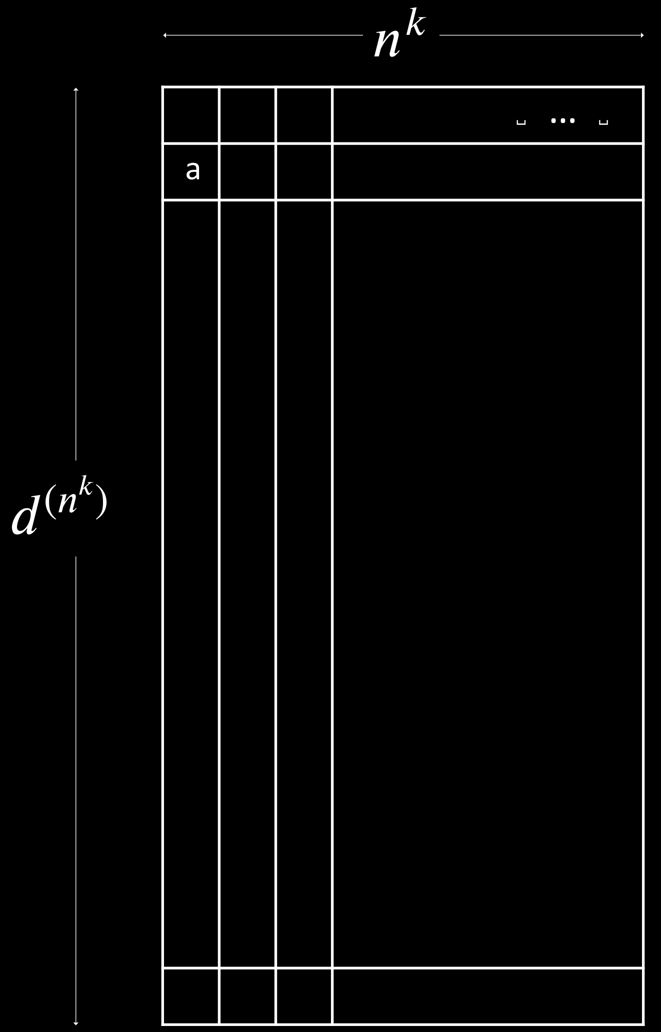
Constructing $\phi_{M,w}$. Try Cook-Levin method.

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



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.

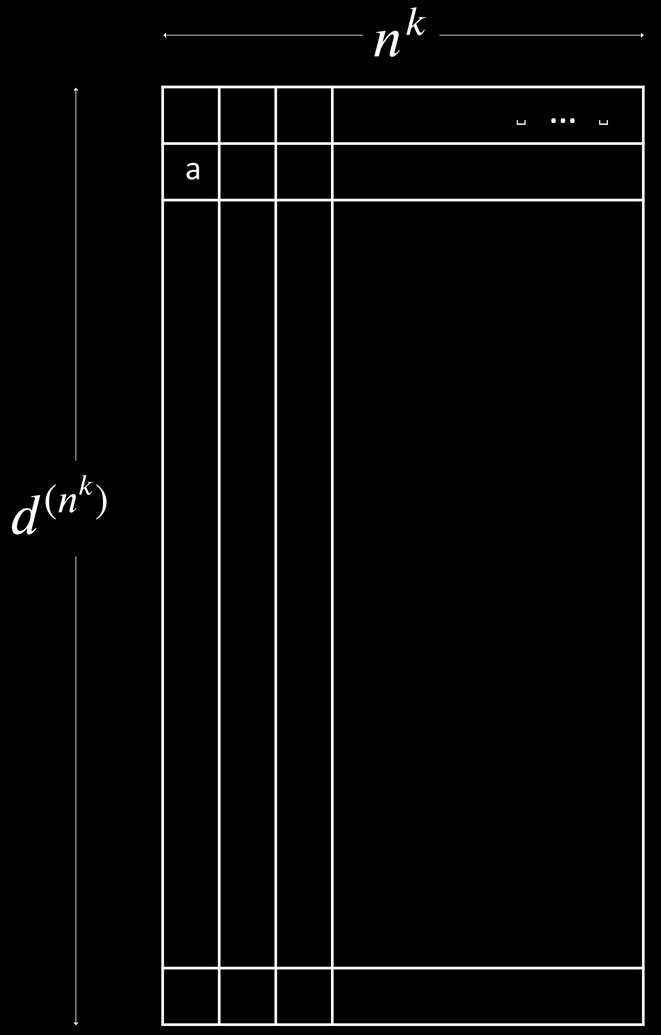
Then $\phi_{M,w}$ will be as big as tableau.

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

Too big! ☹

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

Tableau for M on w



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.

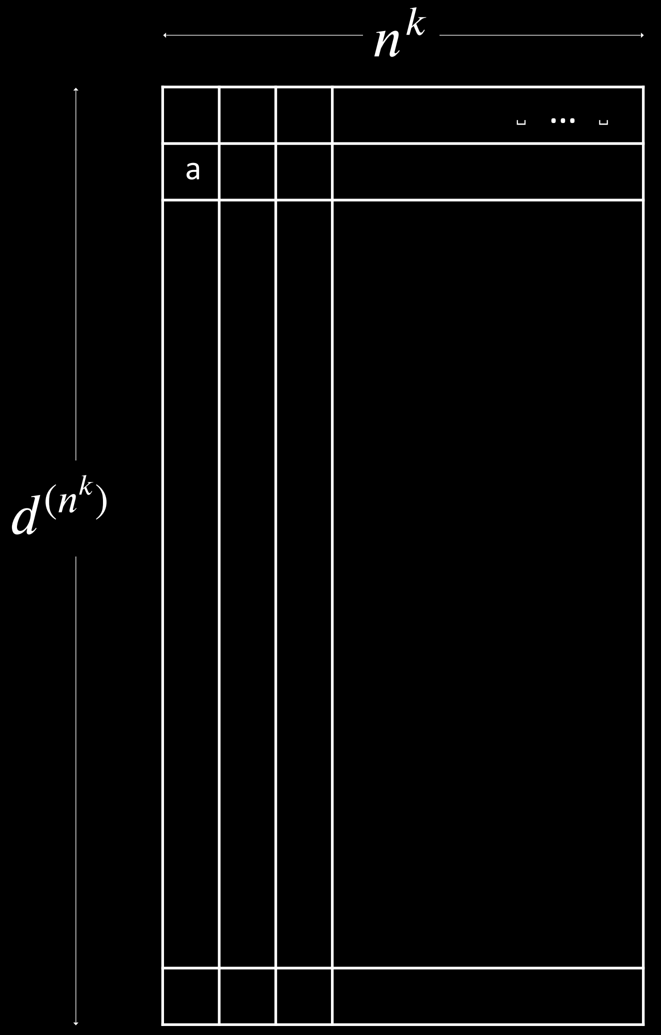
Then $\phi_{M,w}$ will be as big as tableau.

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

Too big! ☹

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

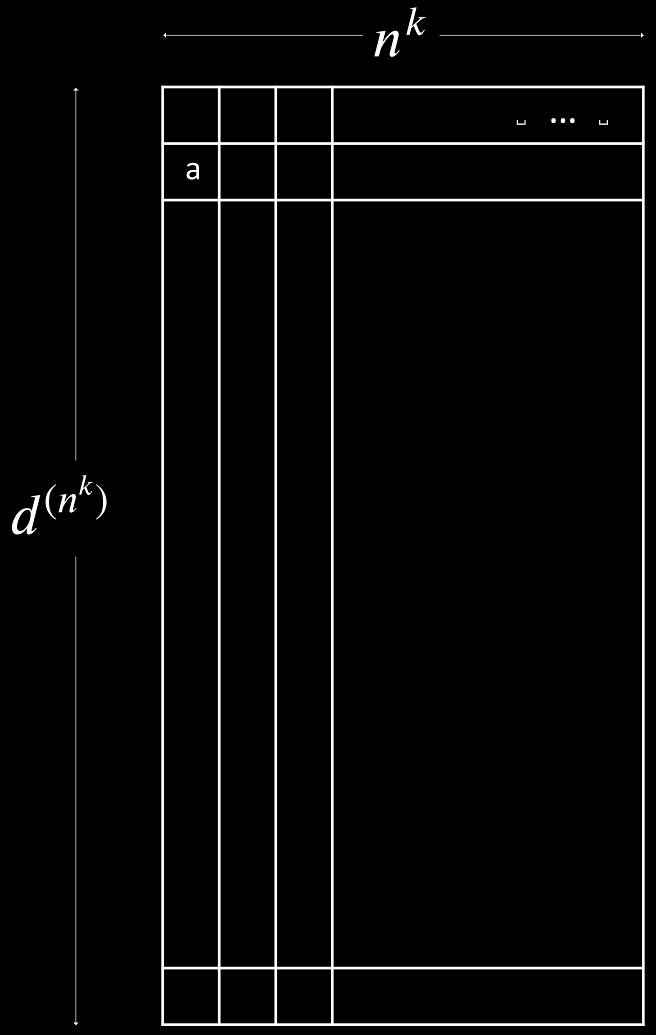
Tableau for M on w



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

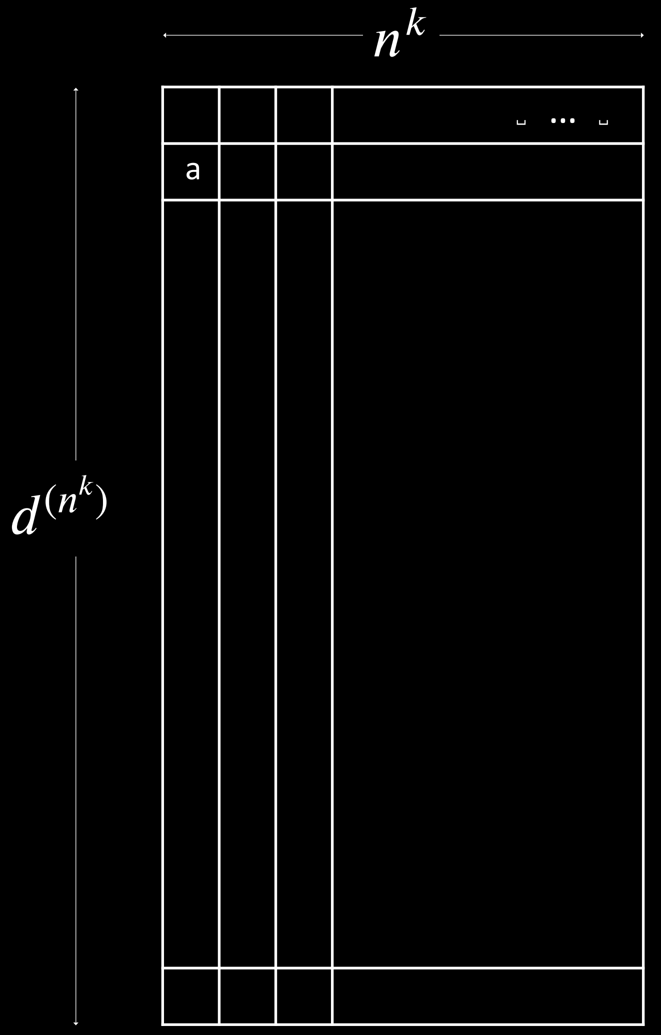
hide →

Tableau for M on w



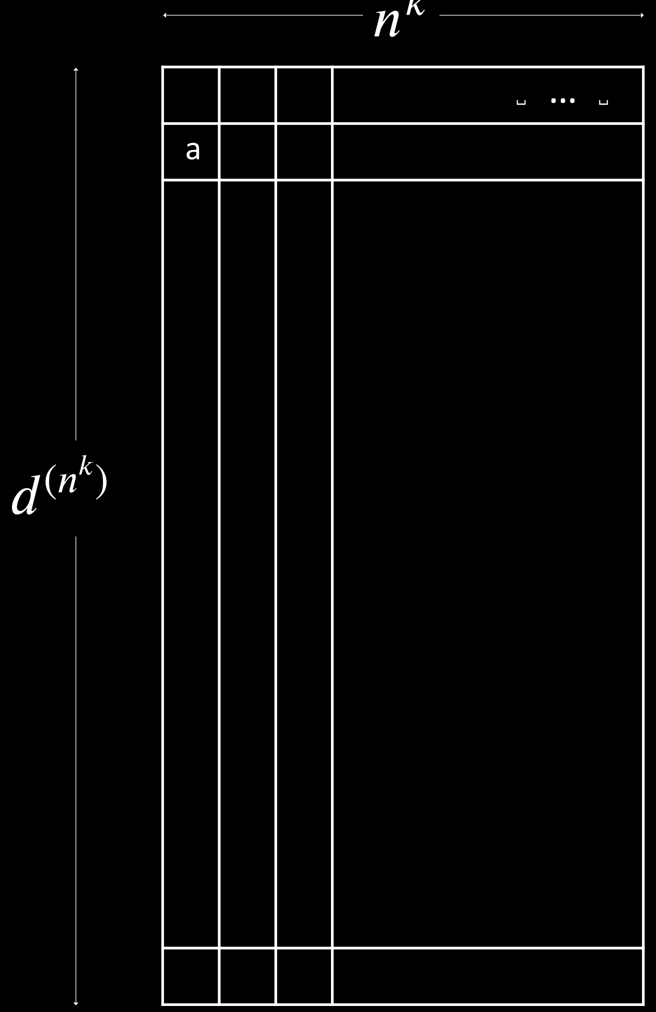
Constructing $\phi_{M,w}$: 2nd try

Tableau for M on w



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

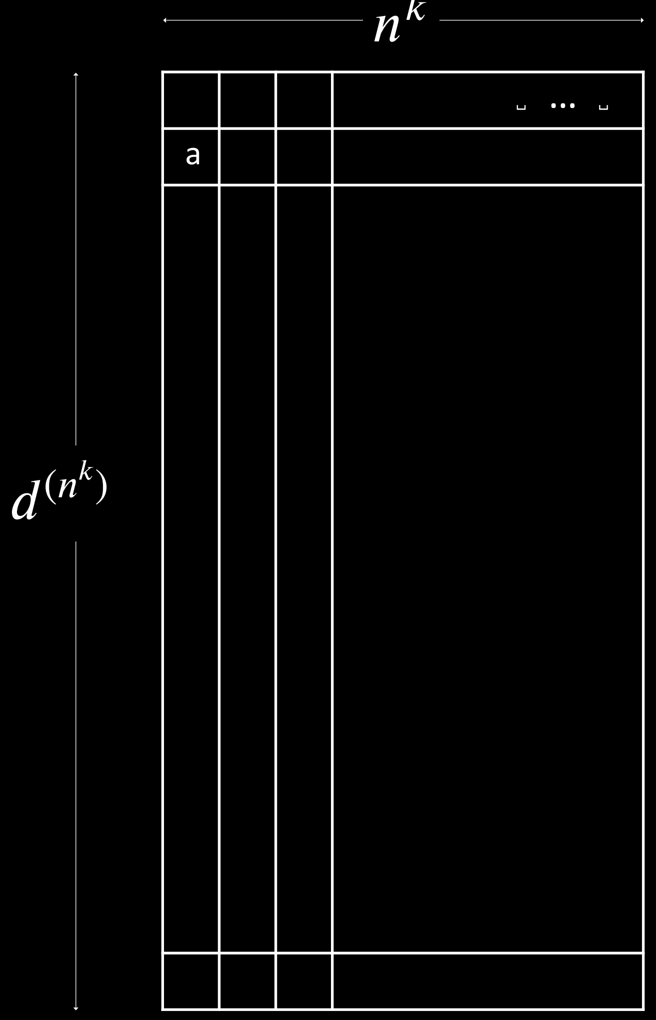
Tableau for M on w



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

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

Tableau for M on w

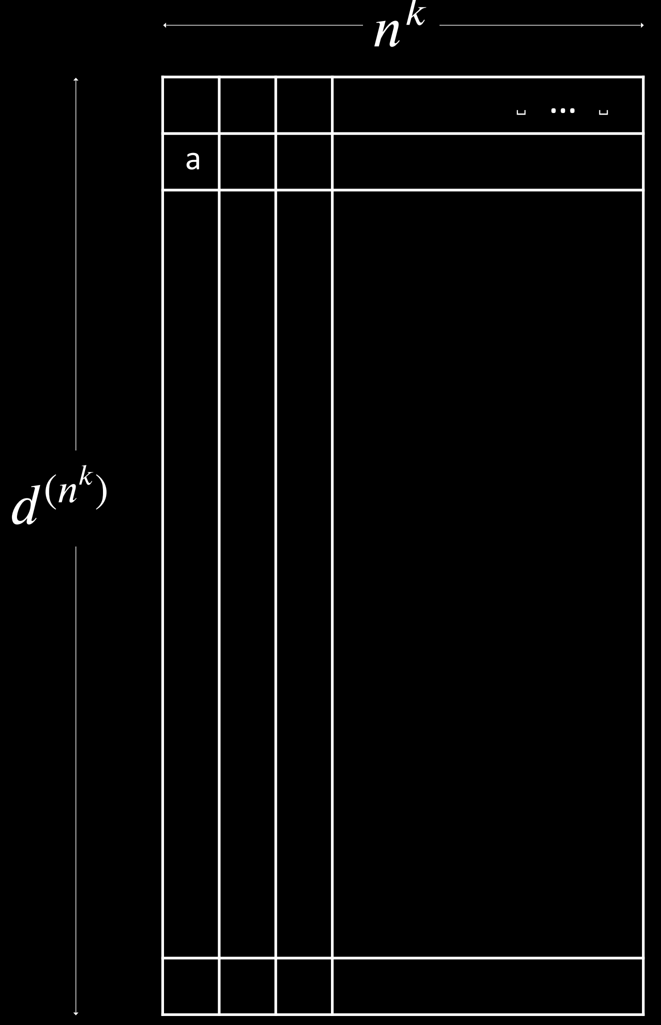


For configs c_i and c_j construct $\phi_{c_i,\ c_j,\ b}$ which "says" $c_i \xrightarrow{b} c_j$ recursively.

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

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

Tableau for M on w

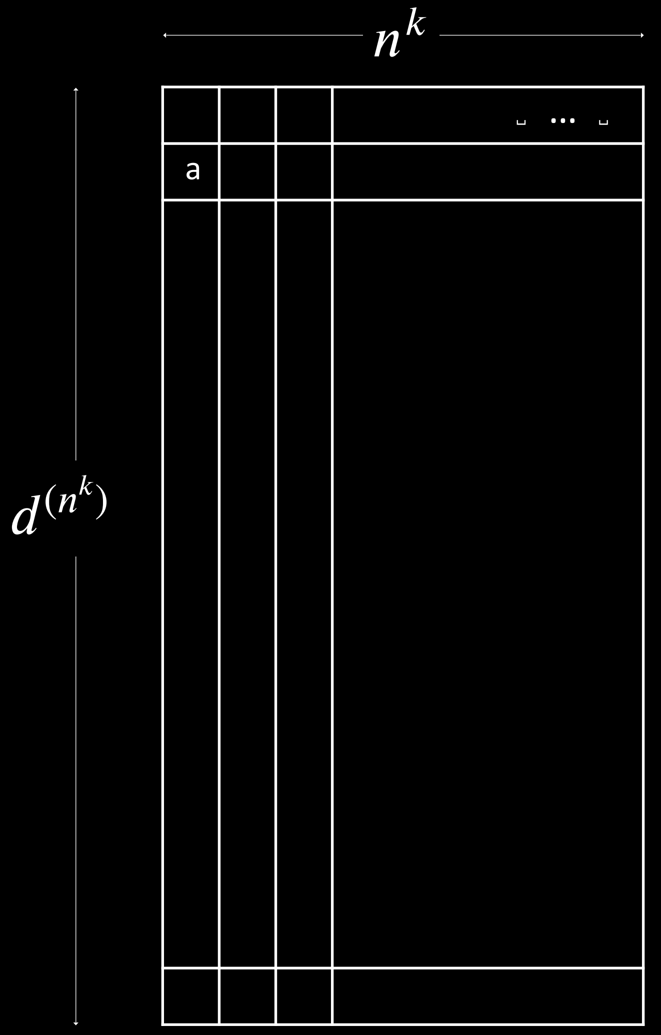


For configs c_i and c_j construct $\phi_{c_i,\ c_j,\ b}$ which "says" $c_i \xrightarrow{b} c_j$ recursively.

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

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

Tableau for M on w



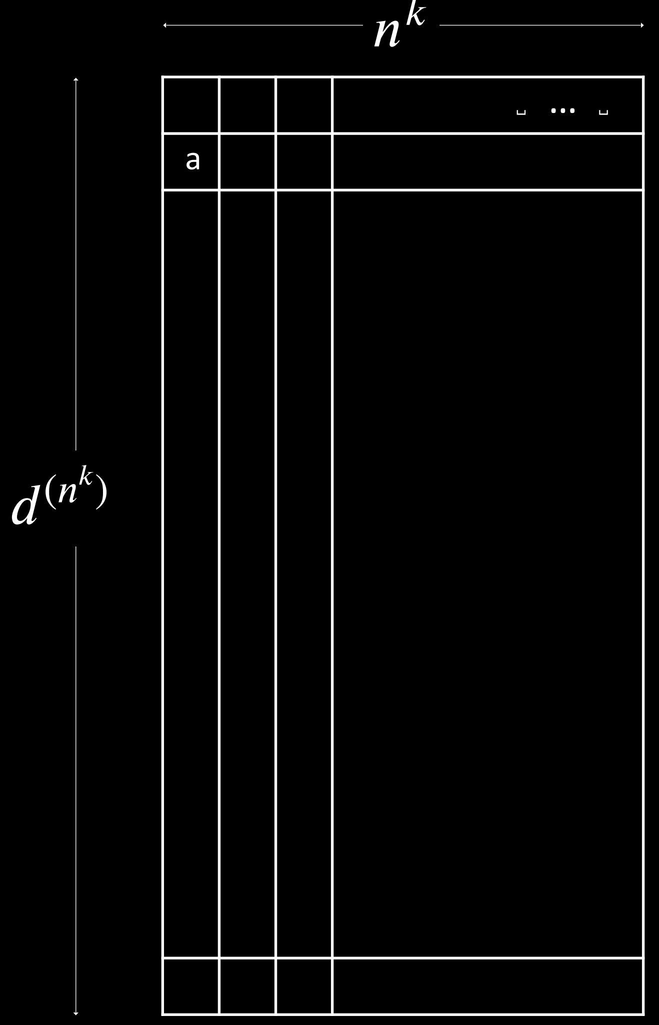
For configs c_i and c_j construct $\phi_{c_i,\ c_j,\ b}$ which "says" $c_i \xrightarrow{b} c_j$ recursively.

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

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

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

Tableau for M on w



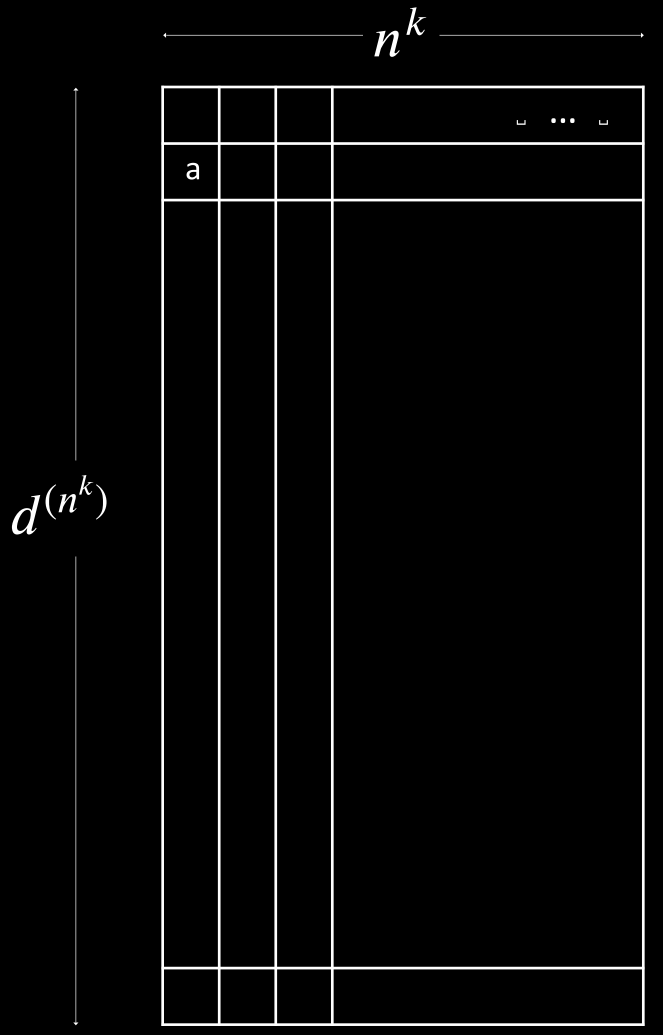
For configs c_i and c_j construct $\phi_{c_i,\ c_j,\ b}$ which "says" $c_i \xrightarrow{b} c_j$ recursively.

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

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

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

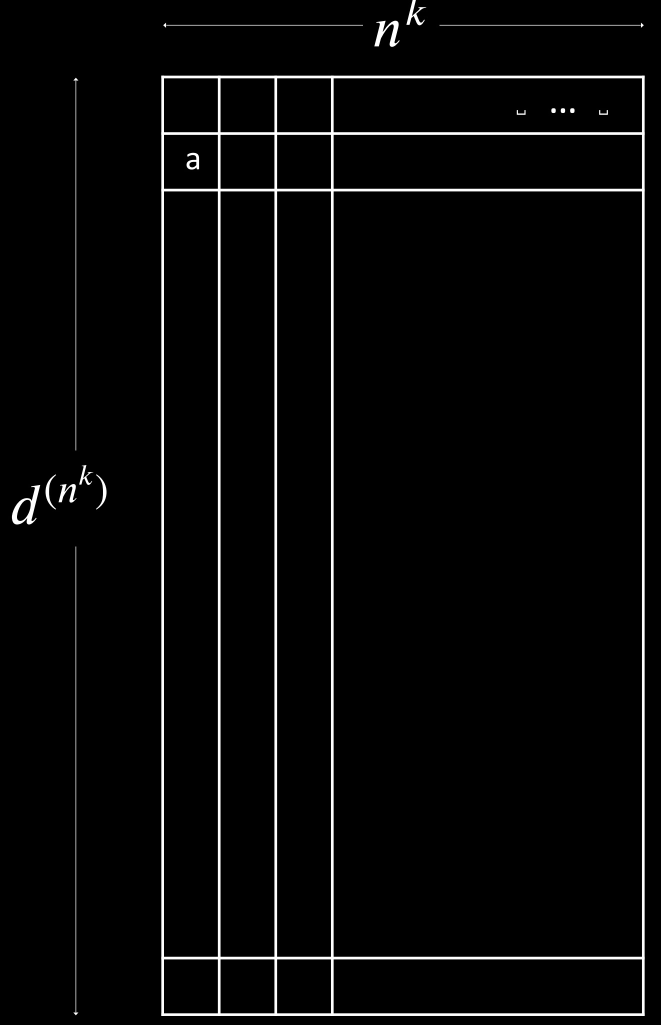
$$\phi_{c_i,\ c_j,\ b} = \underbrace{\exists c_{\mathrm{mid}}}_{\boxed{\substack{\exists x_1, x_2, \cdots, c_l \\ \text{as in Cook-Levin}}}} \left[\phi_{c_i,\ c_{\mathrm{mid}},\ b/2} \ \wedge \ \phi_{c_{\mathrm{mid}},\ c_j,\ b/2} \right]$$

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

$$\exists c_{\mathrm{mid}} [\phi_{\ ,\ ,\ b/8} \cdots]$$

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.

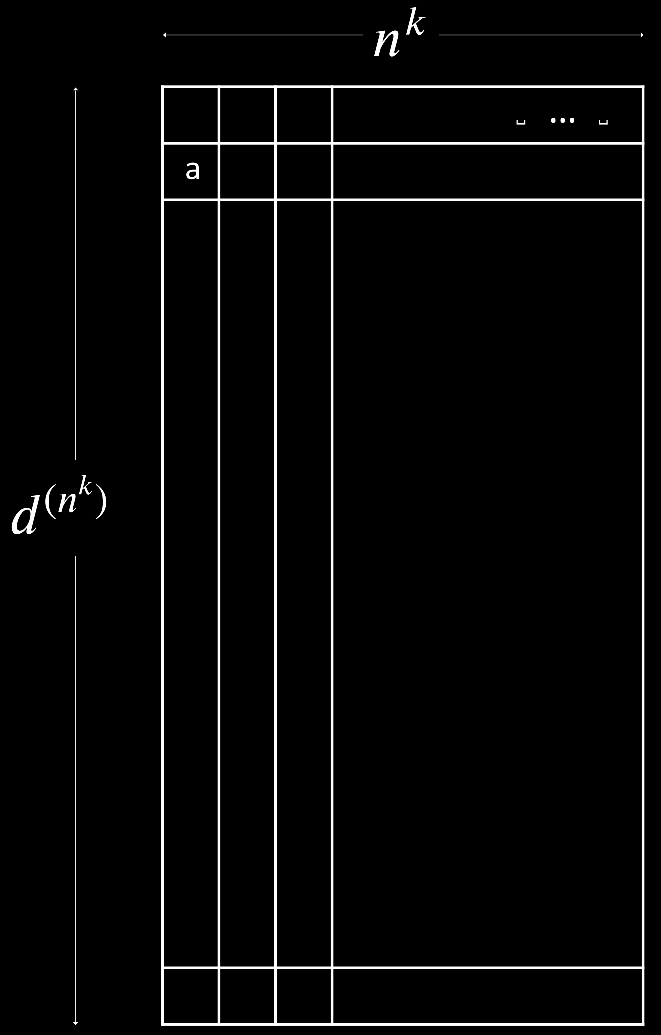
$$\phi_{c_i,\ c_j,\ b} = \underbrace{\exists c_{\text{mid}}}_{\boxed{\substack{\exists x_1, x_2, \cdots, c_l \\ \text{as in Cook-Levin}}}} \left[\phi_{c_i,\, c_{\text{mid}},\ b/2} \ \wedge \ \phi_{c_{\text{mid}},\ c_j,\ b/2} \right]$$

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

$$\vdots \qquad \vdots \qquad \exists c_{\text{mid}} [\phi_{\ ,\ ,\ b/8} \cdots]$$

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.

$$\phi_{c_i,\ c_j,\ b} = \underbrace{\exists c_{\mathrm{mid}}}_{\boxed{\begin{array}{c}\exists x_1, x_2, \cdots, c_l \\ \text{as in Cook-Levin}\end{array}}} \left[\ \phi_{c_i,\ c_{\mathrm{mid}},\ b/2} \ \wedge\ \phi_{c_{\mathrm{mid}},\ c_j,\ b/2}\ \right]$$

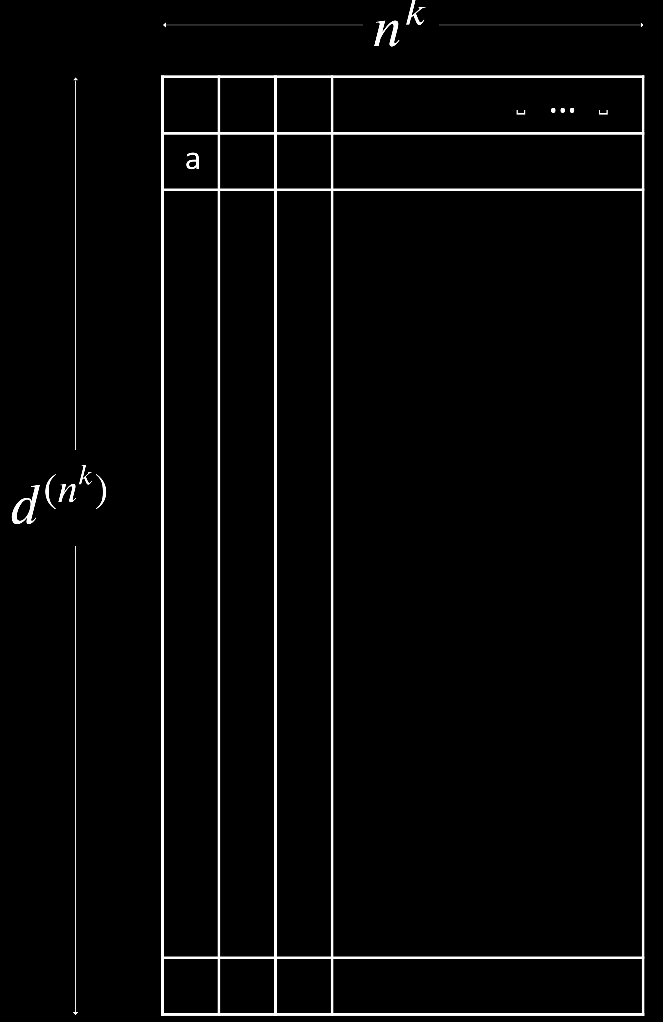
$$\exists c_{\mathrm{mid}}\left[\phi_{\ ,\ ,\ b/4} \ \wedge\ \phi_{\ ,\ ,\ b/4}\right] \qquad \exists c_{\mathrm{mid}}\left[\phi_{\ ,\ ,\ b/4} \ \wedge\ \phi_{\ ,\ ,\ b/4}\right]$$

$$\vdots \qquad\qquad \vdots \qquad\qquad \exists c_{\mathrm{mid}}[\phi_{\ ,\ ,\ b/8} \cdots]$$

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

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.

$$\phi_{c_i,\ c_j,\ b} = \underbrace{\exists c_{\text{mid}}}_{\boxed{\begin{array}{c}\exists x_1, x_2, \cdots, c_l \\ \text{as in Cook-Levin}\end{array}}} \Big[\ \phi_{c_i,\ c_{\text{mid}},\ b/2} \ \wedge \ \phi_{c_{\text{mid}},\ c_j,\ b/2}\ \Big]$$

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

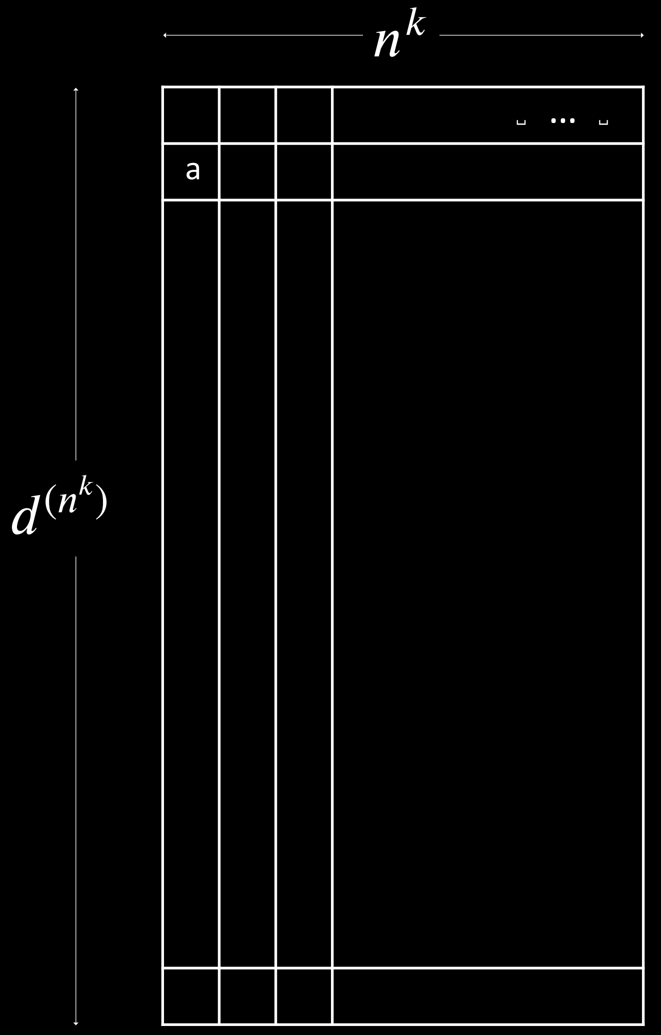
$$\vdots \qquad\qquad \vdots \qquad\qquad \exists c_{\text{mid}}[\phi_{\ ,\ ,\ b/8} \cdots]$$

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

$$\boxed{\begin{array}{c}\phi_{M,w} = \phi_{c_{\text{start}},\ c_{\text{accept}},\ t} \\ t = d^{(n^k)}\end{array}}$$

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.

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

$\exists x_1, x_2, \cdots, c_l$
as in Cook-Levin

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

$\vdots \qquad \vdots \qquad \exists c_{\text{mid}}[\phi_{\ ,\ ,\ b/8} \cdots]$

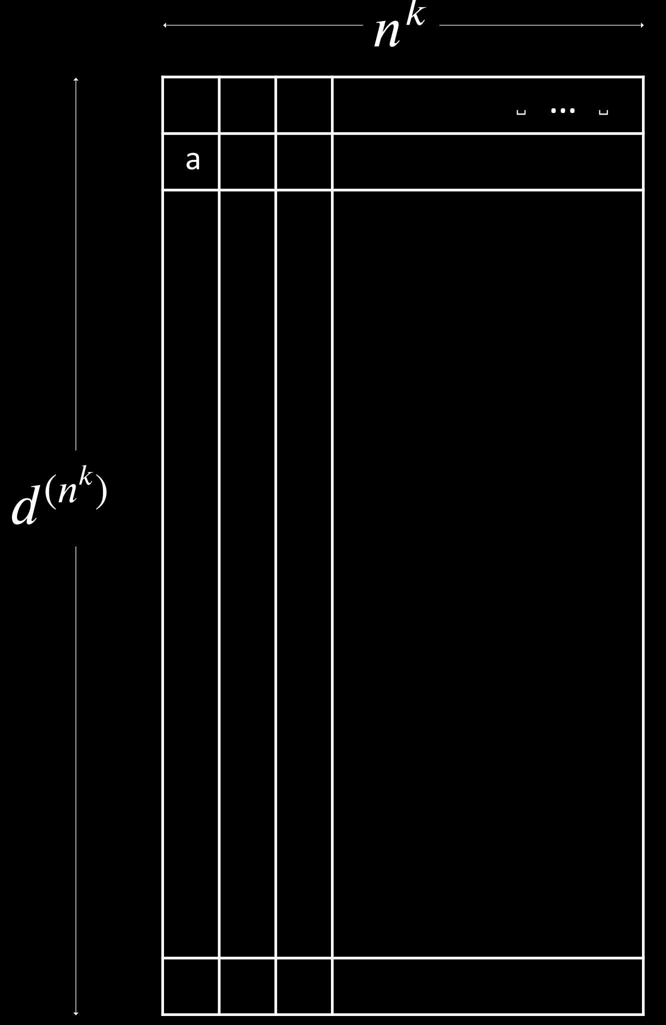
$\phi_{\ ,\ ,\ 1}$ defined as in Cook-Levin

$$\phi_{M,w} = \phi_{c_{\text{start}},\ c_{\text{accept}},\ t}$$
$$t = d^{(n^k)}$$

Size analysis:
Each recursive level doubles number of QBFs.

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

Tableau for M on w



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

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

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

$$\vdots \qquad\qquad \vdots \qquad\qquad \exists c_{\text{mid}}[\phi_{\ ,\ ,\ b/8} \cdots]$$

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

$$\phi_{M,w} = \phi_{c_{\text{start}},\ c_{\text{accept}},\ t}$$
$$t = d^{(n^k)}$$

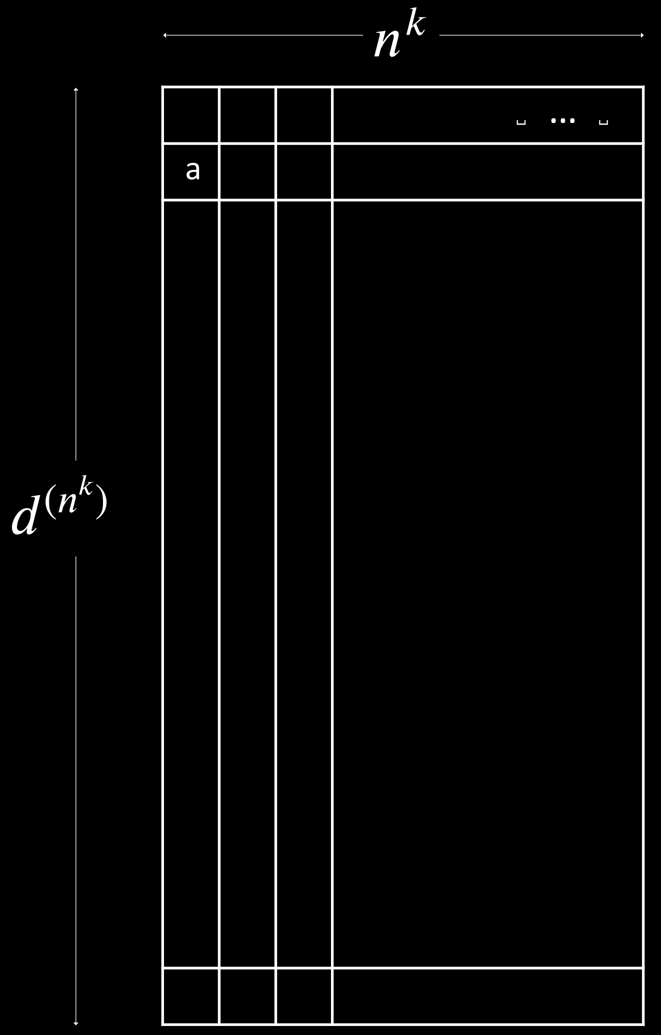
Size analysis:

Each recursive level doubles number of QBFs.

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

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.

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

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

$$\vdots \qquad\qquad \vdots \qquad\qquad \exists c_{\text{mid}}[\phi_{\ ,\ ,\ b/8} \cdots]$$

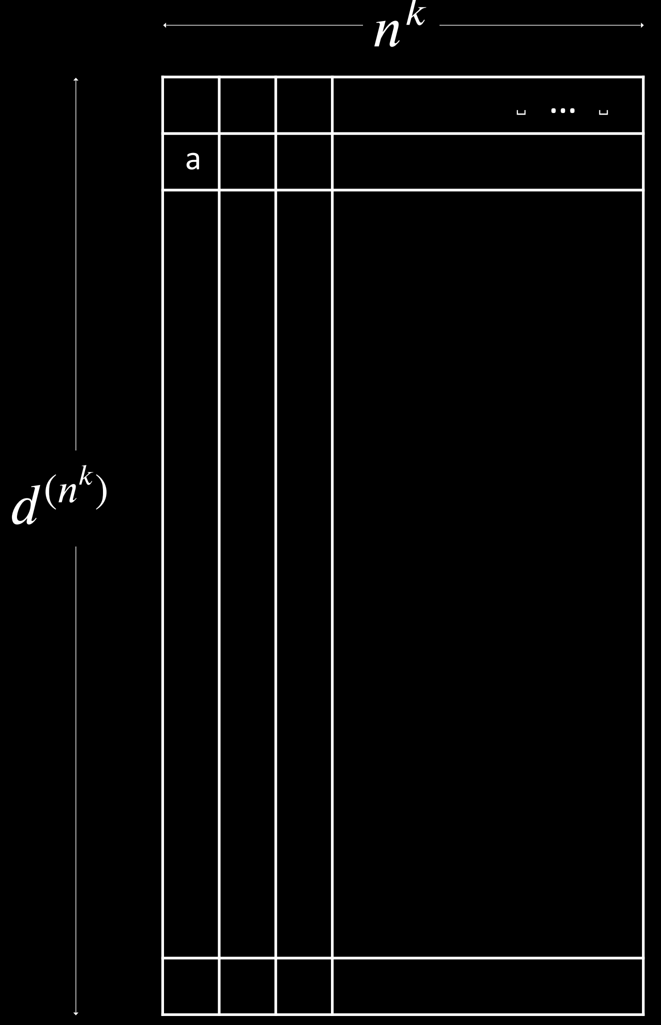
$\phi_{\ ,\ ,\ 1}$ defined as in Cook-Levin

$$\boxed{\begin{gathered}\phi_{M,w} = \phi_{c_{\text{start}},\ c_{\text{accept}},\ t} \\ t = d^{(n^k)}\end{gathered}}$$

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.

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

$\exists x_1, x_2, \cdots, c_l$
as in Cook-Levin

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

$\vdots \qquad \vdots \qquad \exists c_{\text{mid}}[\phi_{\ ,\ ,\ b/8} \cdots]$

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

$$\boxed{\begin{aligned}\phi_{M,w} &= \phi_{c_{\text{start}},\ c_{\text{accept}},\ t} \\ t &= d^{(n^k)}\end{aligned}}$$

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.

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

$\exists x_1, x_2, \cdots, c_l$
as in Cook-Levin

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

$\vdots \qquad \vdots \qquad \exists c_{\text{mid}}[\phi_{\ ,\ ,\ b/8} \cdots]$

$\phi_{\ ,\ ,\ 1}$ 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. ☹

Check-in 18.2

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 $TQBF \notin$ P.

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

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

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

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

$$\vdots$$

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

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

$$\vdots$$

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

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

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

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

$\vdots$

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

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

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

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

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

$\vdots$

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

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

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

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

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

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

$\vdots$

$\forall (x \in S)\ [\,\psi\,]$ is equivalent to $\forall x\ \left[(x \in S) \longrightarrow \psi\right]$

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

$t = d^{\left(n^k\right)}$

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

Size analysis:

Each recursive level <u>adds</u> $O(n^k)$ to the QBF.

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

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

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

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

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

$\vdots$

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

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

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

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

Size analysis:

Each recursive level <u>adds</u> $O(n^k)$ to the QBF.

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

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

Check-in 18.3

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

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

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

$\forall(x \in S)\ [\ \psi\ ]$
is equivalent to
$\forall x\ \left[(x \in S)\ \longrightarrow\ \psi\right]$

$\vdots$

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

$$\phi_{M,w} = \phi_{c_{\text{start}},\ c_{\text{accept}},\ t}$$
$$t = d^{\left(n^k\right)}$$

Size analysis:

Each recursive level <u>adds</u> $O(n^k)$ to the QBF.

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

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

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

Quick review of today

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