COMP3630 / COMP6363

week 10: **Other Complexity Classes**

This Lecture Covers Chapter 11 of HMU: Other Complexity Classes

slides created by: Dirk Pattinson, based on material by
Peter Hoefner and Rob van Glabbeck; with improvements by Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2023

Content of this Chapter

- The classes **PSPACE**, **NPSPACE**, and their co-classes
- The classes **EXPTIME** and **NEXPTIME**
- **PSPACE** vs. **NPSPACE** (Savitch's Theorem)
- Relationship among these (and other) classes

Additional Reading: Chapter 11 of HMU.

Polynomial Space

Definition 11.1.1

A Turing machine M is polyspace bounded if there is a polynomial p so that for all inputs w, M never uses more than $p(|w|)$ tape cells when started with w.

Polynomial Space

Definition 11.1.1

A Turing machine M is polyspace bounded if there is a polynomial p so that for all inputs w, M never uses more than $p(|w|)$ tape cells when started with w.

Note.

- For deterministic machines, this refers to the unique computation path.
- For non-det. machines, this refers to all computation paths starting with input w.

Polynomial Space

Definition 11.1.1

A Turing machine M is polyspace bounded if there is a polynomial p so that for all inputs w, M never uses more than $p(|w|)$ tape cells when started with w.

Note.

- For deterministic machines, this refers to the unique computation path.
- For non-det. machines, this refers to all computation paths starting with input w.

Definition 11.1.2

The class **PSPACE** is the class of languages L such that $L = L(M)$ for a polyspace bounded deterministic Turing machine.

The class **NPSPACE** is the class of languages L such that $L = L(M)$ for a polyspace-bounded non-deterministic Turing machine.

Relationship to Other Classes (A first Look)

Easy Inclusions

P $\subseteq$ **PSPACE** and **NP** $\subseteq$ **NPSPACE**.
(you cannot use more than polynomially many cells in polynomial time, but can spend more time than once on each cell).

Relationship to Other Classes (A first Look)

Easy Inclusions

$\mathbf{P} \subseteq \mathbf{PSPACE}$ and $\mathbf{NP} \subseteq \mathbf{NPSPACE}$.
(you cannot use more than polynomially many cells in polynomial time, but can spend more time than once on each cell).

Inclusions Unknown (to the Literature)

We don't know whether $\mathbf{P} = \mathbf{PSPACE}$ or $\mathbf{NP} = \mathbf{NPSPACE}$ or neither.

Example $\mathsf{ALL}_{\mathsf{NFA}}$

$$\mathsf{ALL}_{\mathsf{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Example $\mathsf{ALL}_{\mathsf{NFA}}$

$$\mathsf{ALL}_{\mathsf{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\mathsf{ALL}_{\mathsf{NFA}} \in$ **NP** nor whether $\mathsf{ALL}_{\mathsf{NFA}} \in$ **co-NP**.
Q. Why don't we know **co-NP**?

Example $\mathsf{ALL}_{\mathsf{NFA}}$

$$\mathsf{ALL}_{\mathsf{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\mathsf{ALL}_{\mathsf{NFA}} \in \mathbf{NP}$ nor whether $\mathsf{ALL}_{\mathsf{NFA}} \in \mathbf{co\text{-}NP}$.
Q. Why don't we know **co-NP**? **A.** Word can be arbitrarily (non-poly) long!

Example $\mathsf{ALL}_{\mathsf{NFA}}$

$$\mathsf{ALL}_{\mathsf{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\mathsf{ALL}_{\mathsf{NFA}} \in \mathbf{NP}$ nor whether $\mathsf{ALL}_{\mathsf{NFA}} \in \mathbf{co\text{-}NP}$.
Q. Why don't we know **co-NP**? **A.** Word can be arbitrarily (non-poly) long!

NPSPACE Algorithm for $\mathsf{ALL}^c_{\mathsf{NFA}}$ – the complement, which accepts A if $L(A) \neq \Sigma^*$

Example $\mathsf{ALL}_{\mathsf{NFA}}$

$$\mathsf{ALL}_{\mathsf{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\mathsf{ALL}_{\mathsf{NFA}} \in$ **NP** nor whether $\mathsf{ALL}_{\mathsf{NFA}} \in$ **co-NP**.
Q. Why don't we know **co-NP**? **A.** Word can be arbitrarily (non-poly) long!

NPSPACE Algorithm for $\mathsf{ALL}^c_{\mathsf{NFA}}$ – the complement, which accepts A if $L(A) \neq \Sigma^*$

Let M implement the following non-deterministic procedure when called with input $\langle A \rangle$ and $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA.

Example ALL_{NFA}

$$\text{ALL}_{\text{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\text{ALL}_{\text{NFA}} \in \mathbf{NP}$ nor whether $\text{ALL}_{\text{NFA}} \in \textbf{co-NP}$.
Q. Why don't we know **co-NP**? **A.** Word can be arbitrarily (non-poly) long!

NPSPACE Algorithm for $\text{ALL}^c_{\text{NFA}}$ – the complement, which accepts A if $L(A) \neq \Sigma^*$

Let M implement the following non-deterministic procedure when called with input $\langle A \rangle$ and $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA.

1. Mark q_0 (as being visited). If $q_0 \notin F$, accept.

Example ALL_{NFA}

$$\text{ALL}_{\text{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\text{ALL}_{\text{NFA}} \in$ **NP** nor whether $\text{ALL}_{\text{NFA}} \in$ **co-NP**.
Q. Why don't we know **co-NP**? **A.** Word can be arbitrarily (non-poly) long!

NPSPACE Algorithm for $\text{ALL}^c_{\text{NFA}}$ – the complement, which accepts A if $L(A) \neq \Sigma^*$

Let M implement the following non-deterministic procedure when called with input $\langle A \rangle$ and $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA.

1. Mark q_0 (as being visited). If $q_0 \notin F$, accept. // Then, $\epsilon \notin L(A)$, thus $L(A) \neq \Sigma^*$

Example $\mathsf{ALL}_{\mathsf{NFA}}$

$$\mathsf{ALL}_{\mathsf{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\mathsf{ALL}_{\mathsf{NFA}} \in$ **NP** nor whether $\mathsf{ALL}_{\mathsf{NFA}} \in$ **co-NP**.
Q. Why don't we know **co-NP**? **A.** Word can be arbitrarily (non-poly) long!

NPSPACE Algorithm for $\mathsf{ALL}^c_{\mathsf{NFA}}$ – the complement, which accepts A if $L(A) \neq \Sigma^*$

Let M implement the following non-deterministic procedure when called with input $\langle A \rangle$ and $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA.

1. Mark q_0 (as being visited). If $q_0 \notin F$, accept. // Then, $\epsilon \notin L(A)$, thus $L(A) \neq \Sigma^*$
2. Repeat $2^{|Q|}$ times:
   1. Let $m \subseteq Q$ be the currently marked states.
   2. Pick some $a \in \Sigma$ and change m to $\bigcup_{q \in m} \delta(q, a)$.
   3. If $m \cap F = \emptyset$, accept.

Example ALL_{NFA}

$$\text{ALL}_{\text{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\text{ALL}_{\text{NFA}} \in$ **NP** nor whether $\text{ALL}_{\text{NFA}} \in$ **co-NP**.
Q. Why don't we know **co-NP**? **A.** Word can be arbitrarily (non-poly) long!

NPSPACE Algorithm for $\text{ALL}^c_{\text{NFA}}$ – the complement, which accepts A if $L(A) \neq \Sigma^*$

Let M implement the following non-deterministic procedure when called with input $\langle A \rangle$ and $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA.

1. Mark q_0 (as being visited). If $q_0 \notin F$, accept. // Then, $\epsilon \notin L(A)$, thus $L(A) \neq \Sigma^*$
2. Repeat $2^{|Q|}$ times:
   1. Let $m \subseteq Q$ be the currently marked states.
   2. Pick some $a \in \Sigma$ and change m to $\bigcup_{q \in m} \delta(q, a)$.
   3. If $m \cap F = \emptyset$, accept. // Then, we found a state that's not accepted.
      // I.e., not all reachable states are accepting states, then some word $wa \notin L(A)$.

Example ALL_{NFA}

$$\text{ALL}_{\text{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\text{ALL}_{\text{NFA}} \in$ **NP** nor whether $\text{ALL}_{\text{NFA}} \in$ **co-NP**.
Q. Why don't we know **co-NP**? **A.** Word can be arbitrarily (non-poly) long!

NPSPACE Algorithm for $\text{ALL}^c_{\text{NFA}}$ – the complement, which accepts A if $L(A) \neq \Sigma^*$

Let M implement the following non-deterministic procedure when called with input $\langle A \rangle$ and $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA.

1. Mark q_0 (as being visited). If $q_0 \notin F$, accept. // Then, $\epsilon \notin L(A)$, thus $L(A) \neq \Sigma^*$
2. Repeat $2^{|Q|}$ times:
   1. Let $m \subseteq Q$ be the currently marked states.
   2. Pick some $a \in \Sigma$ and change m to $\bigcup_{q \in m} \delta(q, a)$.
   3. If $m \cap F = \emptyset$, accept. // Then, we found a state that's not accepted.
      // I.e., not all reachable states are accepting states, then some word $wa \notin L(A)$.
3. reject

Example $\mathrm{ALL}_{\mathrm{NFA}}$

$$\mathrm{ALL}_{\mathrm{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\mathrm{ALL}_{\mathrm{NFA}} \in$ **NP** nor whether $\mathrm{ALL}_{\mathrm{NFA}} \in$ **co-NP**.
Q. Why don't we know **co-NP**? **A.** Word can be arbitrarily (non-poly) long!

NPSPACE Algorithm for $\mathrm{ALL}^c_{\mathrm{NFA}}$ – the complement, which accepts A if $L(A) \neq \Sigma^*$

Let M implement the following non-deterministic procedure when called with input $\langle A \rangle$ and $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA.

1. Mark q_0 (as being visited). If $q_0 \notin F$, accept. // Then, $\epsilon \notin L(A)$, thus $L(A) \neq \Sigma^*$
2. Repeat $2^{|Q|}$ times:
   1. Let $m \subseteq Q$ be the currently marked states.
   2. Pick some $a \in \Sigma$ and change m to $\bigcup_{q \in m} \delta(q, a)$.
   3. If $m \cap F = \emptyset$, accept. // Then, we found a state that's not accepted.
      // I.e., not all reachable states are accepting states, then some word $wa \notin L(A)$.
3. reject // Since we can't find a word that's rejected, so $L(A) = \Sigma^*$

Example ALL_{NFA}

$$\text{ALL}_{\text{NFA}} = \{\langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\text{ALL}_{\text{NFA}} \in$ **NP** nor whether $\text{ALL}_{\text{NFA}} \in$ **co-NP**.
Q. Why don't we know **co-NP**? **A.** Word can be arbitrarily (non-poly) long!

NPSPACE Algorithm for $\text{ALL}^c_{\text{NFA}}$ – the complement, which accepts A if $L(A) \neq \Sigma^*$

Let M implement the following non-deterministic procedure when called with input $\langle A \rangle$ and $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA.

1. Mark q_0 (as being visited). If $q_0 \notin F$, accept. // Then, $\epsilon \notin L(A)$, thus $L(A) \neq \Sigma^*$
2. Repeat $2^{|Q|}$ times:
   1. Let $m \subseteq Q$ be the currently marked states.
   2. Pick some $a \in \Sigma$ and change m to $\bigcup_{q \in m} \delta(q, a)$.
   3. If $m \cap F = \emptyset$, accept. // Then, we found a state that's not accepted.
      // I.e., not all reachable states are accepting states, then some word $wa \notin L(A)$.
3. reject // Since we can't find a word that's rejected, so $L(A) = \Sigma^*$

- M may use exponential time but linear space only.

Example ALL_{NFA}

$$\text{ALL}_{\text{NFA}} = \{\langle A\rangle : A \text{ is an NFA and } L(A) = \Sigma^*\}$$

Currently, it's known neither whether $\text{ALL}_{\text{NFA}} \in$ **NP** nor whether $\text{ALL}_{\text{NFA}} \in$ **co-NP**.
Q. Why don't we know **co-NP**? **A.** Word can be arbitrarily (non-poly) long!

NPSPACE Algorithm for $\text{ALL}^c_{\text{NFA}}$ – the complement, which accepts A if $L(A) \neq \Sigma^*$

Let M implement the following non-deterministic procedure when called with input $\langle A\rangle$ and $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA.

1. Mark q_0 (as being visited). If $q_0 \notin F$, accept. // Then, $\epsilon \notin L(A)$, thus $L(A) \neq \Sigma^*$
2. Repeat $2^{|Q|}$ times:
   1. Let $m \subseteq Q$ be the currently marked states.
   2. Pick some $a \in \Sigma$ and change m to $\bigcup_{q\in m} \delta(q, a)$.
   3. If $m \cap F = \emptyset$, accept. // Then, we found a state that's not accepted.
      // I.e., not all reachable states are accepting states, then some word $wa \notin L(A)$.
3. reject // Since we can't find a word that's rejected, so $L(A) = \Sigma^*$

- M may use exponential time but linear space only.
- Hence $\text{ALL}^c_{\text{NFA}} \in$ **NPSPACE** – and thus, by definition, $\text{ALL}_{\text{NFA}} \in$ **co-NPSPACE**

PSPACE vs. **co-PSPACE**

Theorem 11.1.3

PSPACE $=$ **co-PSPACE** *(and* **NPSPACE** $=$ **co-NPSPACE***)*

Proof.

- Let $L \in$ **PSPACE** (resp., $L \in$ **co-PSPACE**).
- Decide L^c in **PSPACE** (resp., $L^c \in$ **co-PSPACE**) via:
  - First, decide $L \in$ **PSPACE** (resp., $L \in$ **co-PSPACE**).
  - Then, flip result. This decides L^c, taking poly-space.
- Intuitively, there's no reason why result flipping should not be allowed in a space class.

The same arguments work for **NPSPACE**/**co-NPSPACE**. □

PSPACE vs. **co-PSPACE**

Theorem 11.1.3

PSPACE = **co-PSPACE** *(and* **NPSPACE** = **co-NPSPACE***)*

Proof.

- Let $L \in$ **PSPACE** (resp., $L \in$ **co-PSPACE**).
- Decide L^c in **PSPACE** (resp., $L^c \in$ **co-PSPACE**) via:
  - First, decide $L \in$ **PSPACE** (resp., $L \in$ **co-PSPACE**).
  - Then, flip result. This decides L^c, taking poly-space.
- Intuitively, there's no reason why result flipping should not be allowed in a space class.

The same arguments work for **NPSPACE**/**co-NPSPACE**. □

Note on ALL_{NFA}

- Later, we will show that **PSPACE** = **NPSPACE**.
- Thus, $\text{ALL}_{\text{NFA}} \in$ **PSPACE**.

Exponential Time

Definition 11.2.1

A deterministic or non-deterministic Turing machine runs in exponential time if it terminates in at most $c^{p(|w|)}$ steps for a constant c and polynomial p.

EXPTIME is the class of languages L for which $L = L(M)$ for an exptime deterministic Turing machine.

NEXPTIME is the class of languages L for which $L = L(M)$ for a nondeterministic exponential time Turing machine.

(More) Easy Inclusions

Recap:

- $\mathbf{P} \subseteq \mathbf{PSPACE}$ and $\mathbf{NP} \subseteq \mathbf{NPSPACE}$.
- $\mathbf{PSPACE} = \mathbf{co\text{-}PSPACE}$ and $\mathbf{NPSPACE} = \mathbf{co\text{-}NPSPACE}$

Exponential Time

Definition 11.2.1

A deterministic or non-deterministic Turing machine runs in exponential time if it terminates in at most $c^{p(|w|)}$ steps for a constant c and polynomial p.

EXPTIME is the class of languages L for which $L = L(M)$ for an exptime deterministic Turing machine.

NEXPTIME is the class of languages L for which $L = L(M)$ for a nondeterministic exponential time Turing machine.

(More) Easy Inclusions

Recap:

- **P** $\subseteq$ **PSPACE** and **NP** $\subseteq$ **NPSPACE**.
- **PSPACE** $=$ **co-PSPACE** and **NPSPACE** $=$ **co-NPSPACE**

Now also:

- **EXPTIME** $\subseteq$ **NEXPTIME**
- **P** $\subsetneq$ **EXPTIME**, that's one of the very few inclusions known to be proper

Exponential Time

Definition 11.2.1

A deterministic or non-deterministic Turing machine runs in exponential time if it terminates in at most $c^{p(|w|)}$ steps for a constant c and polynomial p.

EXPTIME is the class of languages L for which $L = L(M)$ for an exptime deterministic Turing machine.

NEXPTIME is the class of languages L for which $L = L(M)$ for a nondeterministic exponential time Turing machine.

(More) Easy Inclusions

Recap:

- **P** $\subseteq$ **PSPACE** and **NP** $\subseteq$ **NPSPACE**.
- **PSPACE** $=$ **co-PSPACE** and **NPSPACE** $=$ **co-NPSPACE**

Now also:

- **EXPTIME** $\subseteq$ **NEXPTIME**
- **P** $\subsetneq$ **EXPTIME**, that's one of the very few inclusions known to be proper

Still to show: **PSPACE** $\subseteq$ **EXPTIME** (not that easy, but not that hard either)

PSPACE vs. **EXPTIME**

Theorem 11.2.2

PSPACE $\subseteq$ **EXPTIME**

Proof.

- Let $L \in$ **PSPACE**.

PSPACE vs. **EXPTIME**

Theorem 11.2.2

PSPACE $\subseteq$ **EXPTIME**

Proof.

- Let $L \in$ **PSPACE**.
- Then, L is decided by some TM M, such that for all w it decides $w \in L$ with $|w| = n$ within $O(n^k)$ space for some constant k.
- How many different TM configurations can we see before running into a loop?

PSPACE vs. **EXPTIME**

Theorem 11.2.2

PSPACE $\subseteq$ **EXPTIME**

Proof.

- Let $L \in$ **PSPACE**.
- Then, L is decided by some TM M, such that for all w it decides $w \in L$ with $|w| = n$ within $O(n^k)$ space for some constant k.
- How many different TM configurations can we see before running into a loop?
  - Each cell can have at most $|\Gamma|$ different symbols.
  - So we have at most $O(|\Gamma|^{(n^k)})$ different tape configurations.

PSPACE vs. EXPTIME

Theorem 11.2.2

PSPACE $\subseteq$ **EXPTIME**

Proof.

- Let $L \in$ **PSPACE**.
- Then, L is decided by some TM M, such that for all w it decides $w \in L$ with $|w| = n$ within $O(n^k)$ space for some constant k.
- How many different TM configurations can we see before running into a loop?
  - Each cell can have at most $|\Gamma|$ different symbols.
  - So we have at most $O(|\Gamma|^{(n^k)})$ different tape configurations.
  - We have $|Q|$ states and at most $O(n^k)$ head positions.
  - In total we have at most $c^{p(n)} = O(|Q| \cdot (n^k) \cdot |\Gamma|^{(n^k)})$ TM configurations.

PSPACE vs. EXPTIME

Theorem 11.2.2

PSPACE $\subseteq$ **EXPTIME**

Proof.

- Let $L \in$ **PSPACE**.
- Then, L is decided by some TM M, such that for all w it decides $w \in L$ with $|w| = n$ within $O(n^k)$ space for some constant k.
- How many different TM configurations can we see before running into a loop?
  - Each cell can have at most $|\Gamma|$ different symbols.
  - So we have at most $O(|\Gamma|^{(n^k)})$ different tape configurations.
  - We have $|Q|$ states and at most $O(n^k)$ head positions.
  - In total we have at most $c^{p(n)} = O(|Q| \cdot (n^k) \cdot |\Gamma|^{(n^k)})$ TM configurations.
- Since k is a constant, we need at most exponential time before running into a loop (which we don't have to since the problem is decided).

$\square$

Savitch's Theorem: **PSPACE** = **NPSPACE**

Note

The following is (maybe?) remarkable because we do not know whether **P** = **NP**.

Theorem 11.3.1

PSPACE = **NPSPACE** *Savitch's Theorem, 1970*

Savitch's Theorem: **PSPACE** $=$ **NPSPACE**

Note

The following is (maybe?) remarkable because we do not know whether **P** $=$ **NP**.

Theorem 11.3.1

PSPACE $=$ **NPSPACE** *Savitch's Theorem, 1970*

Proof.

- Let $L \in$ **NPSPACE** and M be non-det. TM, polyspace-bounded by $p(n)$ deciding L.

Savitch's Theorem: **PSPACE** = **NPSPACE**

Note

The following is (maybe?) remarkable because we do not know whether **P** = **NP**.

Theorem 11.3.1

PSPACE = **NPSPACE** *Savitch's Theorem, 1970*

Proof.

- Let $L \in$ **NPSPACE** and M be non-det. TM, polyspace-bounded by $p(n)$ deciding L.
- Noteworthy[1], we are allowed to assume that M has the following properties:
  - M has just a single accepting state, which is a halting state.

[1] (Related to why we were allowed to assume that our CFL is given in Chomsky NF, cf. Theorem 10.2.9.)

Savitch's Theorem: **PSPACE** = **NPSPACE**

Note

The following is (maybe?) remarkable because we do not know whether **P** = **NP**.

Theorem 11.3.1

PSPACE = **NPSPACE** *Savitch's Theorem, 1970*

Proof.

- Let $L \in$ **NPSPACE** and M be non-det. TM, polyspace-bounded by $p(n)$ deciding L.
- Noteworthy[1], we are allowed to assume that M has the following properties:
  - M has just a single accepting state, which is a halting state.
  - When it accepts, the tape is empty.
  - Taken together, there is just a single halting configuration. (We call it J.)

[1](Related to why we were allowed to assume that our CFL is given in Chomsky NF, cf. Theorem 10.2.9.)

Savitch's Theorem: PSPACE = NPSPACE

Note

The following is (maybe?) remarkable because we do not know whether **P** = **NP**.

Theorem 11.3.1

PSPACE = **NPSPACE** *Savitch's Theorem, 1970*

Proof.

- Let $L \in$ **NPSPACE** and M be non-det. TM, polyspace-bounded by $p(n)$ deciding L.
- Noteworthy[1], we are allowed to assume that M has the following properties:
  - M has just a single accepting state, which is a halting state.
  - When it accepts, the tape is empty.
  - Taken together, there is just a single halting configuration. (We call it J.)
- Recall that M has $c^{p(n)}$ different IDs, were $n = |w|$.

[1] (Related to why we were allowed to assume that our CFL is given in Chomsky NF, cf. Theorem 10.2.9.)

Savitch's Theorem: **PSPACE** = **NPSPACE**

Note

The following is (maybe?) remarkable because we do not know whether **P** = **NP**.

Theorem 11.3.1

PSPACE = **NPSPACE** *Savitch's Theorem, 1970*

Proof.

- Let $L \in$ **NPSPACE** and M be non-det. TM, polyspace-bounded by $p(n)$ deciding L.
- Noteworthy[1], we are allowed to assume that M has the following properties:
  - M has just a single accepting state, which is a halting state.
  - When it accepts, the tape is empty.
  - Taken together, there is just a single halting configuration. (We call it J.)
- Recall that M has $c^{p(n)}$ different IDs, were $n = |w|$.
- Design a deterministic TM M', which decides whether $I \vdash^* J$ is possible within at most $c^{p(n)}$ steps. M' is space-bounded by $p(n)$.

[1](Related to why we were allowed to assume that our CFL is given in Chomsky NF, cf. Theorem 10.2.9.)

Savitch's Theorem: **PSPACE** = **NPSPACE**

Note

The following is (maybe?) remarkable because we do not know whether **P** = **NP**.

Theorem 11.3.1

PSPACE = **NPSPACE** *Savitch's Theorem, 1970*

Proof.

- Let $L \in$ **NPSPACE** and M be non-det. TM, polyspace-bounded by $p(n)$ deciding L.
- Noteworthy[1], we are allowed to assume that M has the following properties:
  - M has just a single accepting state, which is a halting state.
  - When it accepts, the tape is empty.
  - Taken together, there is just a single halting configuration. (We call it J.)
- Recall that M has $c^{p(n)}$ different IDs, were $n = |w|$.
- Design a deterministic TM M', which decides whether $I \vdash^* J$ is possible within at most $c^{p(n)}$ steps. M' is space-bounded by $p(n)$.
- We formalize this via predicate $P(ID_1, ID_2, m)$, initialized to $P(I, J, c^{p(n)})$. □

[1](Related to why we were allowed to assume that our CFL is given in Chomsky NF, cf. Theorem 10.2.9.)

Savitch's Theorem: Recursive Doubling

Goal. Implement $P(I, J, m) = I \vdash^* J$ in deterministic polyspace

```
P(I, J, m): for all IDs K with length <= p(n) do {
  if P(I, K, m/2) and P(K, J, m/2) then return true
}
return false
```

Q. How much space does this implementation need? (Time does not matter!)

Savitch's Theorem: Recursive Doubling

Goal. Implement $P(I, J, m) = I \vdash^* J$ in deterministic polyspace

```
P(I, J, m): for all IDs K with length <= p(n) do {
  if P(I, K, m/2) and P(K, J, m/2) then return true
}
return false
```

Q. How much space does this implementation need? (Time does not matter!)

$$P(I, K_0 = J, m)$$

$$P(I, K_1, {}^{m}\!/\!_{2}) \qquad P(K_1, K_0 = J, {}^{m}\!/\!_{2})$$

$$P(I, K_2, {}^{m}\!/\!_{4}) \quad P(K_2, K_1 {}^{m}\!/\!_{4})$$

$$\ldots \qquad \ldots$$

$$P(I, K_i, {}^{m}\!/\!_{2^i}) \quad P(K_i, K_{i-1}, {}^{m}\!/\!_{2^i})$$

Savitch's Theorem: Recursive Doubling

Goal. Implement $P(I, J, m) = I \vdash^* J$ in deterministic polyspace

```
P(I, J, m): for all IDs K with length <= p(n) do {
  if P(I, K, m/2) and P(K, J, m/2) then return true
}
return false
```

Q. How much space does this implementation need? (Time does not matter!)

$P(I, K_0 = J, m)$

$P(I, K_1, {}^{m}/_{2})$ $\quad$ $P(K_1, K_0 = J, {}^{m}/_{2})$

$P(I, K_2, {}^{m}/_{4})$ $\quad$ $P(K_2, K_1 {}^{m}/_{4})$

$\ldots$ $\quad$ $\ldots$

$P(I, K_i, {}^{m}/_{2^i})$ $\quad$ $P(K_i, K_{i-1}, {}^{m}/_{2^i})$

- Required space: $\mathcal{O}(log(c^{p(n)}) \cdot p(n)) = \mathcal{O}(p^2(n))$.

Savitch's Theorem: Recursive Doubling

Goal. Implement $P(I, J, m) = I \vdash^* J$ in deterministic polyspace

```
P(I, J, m): for all IDs K with length <= p(n) do {
  if P(I, K, m/2) and P(K, J, m/2) then return true
}
return false
```

Q. How much space does this implementation need? (Time does not matter!)

$$P(I, K_0 = J, m)$$

$$P(I, K_1, {}^{m}/_{2}) \qquad P(K_1, K_0 = J, {}^{m}/_{2})$$

$$P(I, K_2, {}^{m}/_{4}) \qquad P(K_2, K_1 {}^{m}/_{4})$$

$$\ldots \qquad \ldots$$

$$P(I, K_i, {}^{m}/_{2^i}) \qquad P(K_i, K_{i-1}, {}^{m}/_{2^i})$$

- Required space: $\mathcal{O}(log(c^{p(n)}) \cdot p(n)) = \mathcal{O}(p^2(n))$.

Q. Earlier we were assuming that there's a unique J. Did we have to?

Savitch's Theorem: Recursive Doubling

Goal. Implement $P(I, J, m) = I \vdash^* J$ in deterministic polyspace

```
P(I, J, m): for all IDs K with length <= p(n) do {
  if P(I, K, m/2) and P(K, J, m/2) then return true
}
return false
```

Q. How much space does this implementation need? (Time does not matter!)

$$P(I, K_0 = J, m)$$

$$P(I, K_1, {}^m/_2) \qquad P(K_1, K_0 = J, {}^m/_2)$$

$$P(I, K_2, {}^m/_4) \quad P(K_2, K_1 {}^m/_4)$$

$$\ldots \qquad \ldots$$

$$P(I, K_i, {}^m/_{2^i}) \quad P(K_i, K_{i-1}, {}^m/_{2^i})$$

- Required space: $\mathcal{O}(log(c^{p(n)}) \cdot p(n)) = \mathcal{O}(p^2(n))$.

Q. Earlier we were assuming that there's a unique J. Did we have to?

A. No, we could have just generated all possible (accepting) IDs and try all of them!

Relationship to Other Classes (Recap)

(Some of the) Classes covered so far

$$\mathbf{P} \neq \mathbf{EXPTIME} \tag{1}$$

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{NEXPTIME} \tag{2}$$

$$\mathbf{co\text{-}PSPACE} = \mathbf{PSPACE} = \mathbf{NPSPACE} = \mathbf{co\text{-}NPSPACE} \tag{3}$$

Note:

- Relationships of the other co-classes for time are not shown.
- In (2), at least one inclusion must be proper (see (1)!), but we don't know which!
- There are still *many* more classes,
  - both on the right (**(N)EXPSPACE**, **DEXPTIME**, ...),
  - in between, and
  - there are even classes of infinitely large hierarchies.