Computational Complexity Theory

Lecture 10: Space complexity classes

Department of Computer Science,
Indian Institute of Science

Space bounded computation

- Here, we are interested to find out how much of *work space* is required to solve a problem.
- For convenience, think of TMs with a separate read-only input tape and one or more work tapes. Work space is the number of cells in the work tapes of a TM M visited by M's heads during a computation.

Space bounded computation

- Here, we are interested to find out how much of *work space* is required to solve a problem.
- For convenience, think of TMs with a separate read-only input tape and one or more work tapes. Work space is the number of cells in the work tapes of a TM M visited by M's heads during a computation.

- Definition. Let $S: \mathbb{N} \rightarrow \mathbb{N}$ be a function. A language L is in $\mathrm{DSPACE}(S(n))$ if there's a TM M that decides L using $O(S(n))$ work space on inputs of length n.

Space bounded computation

- Here, we are interested to find out how much of *work space* is required to solve a problem.
- For convenience, think of TMs with a separate read-only input tape and one or more work tapes. Work space is the number of cells in the work tapes of a TM M visited by M's heads during a computation.

- Definition. Let S: $\mathbb{N} \rightarrow \mathbb{N}$ be a function. A language L is in NSPACE(S(n)) if there's a NTM M that decides L using O(S(n)) work space on inputs of length n, regardless of M's nondeterministic choices.

Space bounded computation

- We'll refer to 'work space' as 'space'. For convenience, assume there's a single work tape.
- If the output has many bits, then we will assume that the TM has a separate write-only output tape.

Space bounded computation

- We'll refer to 'work space' as 'space'. For convenience, assume there's a <u>single</u> work tape.
- If the output has many bits, then we will assume that the TM has a separate write-only <u>output tape</u>.

- Definition. Let $S: \mathbb{N} \longrightarrow \mathbb{N}$ be a function. S is <u>*space constructible*</u> if $S(n) \geq \log n$ and there's a TM that computes $S(|x|)$ from x using $O(S(|x|))$ space.

Relation between time and space

- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

Hopcroft, Paul & Valiant 1977

Relation between time and space

- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, if S is space constructible.
- Proof. Uses the notion of *configuration graph* of a TM. We'll see this shortly.

Relation between time and space

- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, if S is space constructible.

- Definition.

$$\begin{aligned} \mathrm{L} &= \mathrm{DSPACE}(\log n) \\ \mathrm{NL} &= \mathrm{NSPACE}(\log n) \\ \mathrm{PSPACE} &= \bigcup_{c>0} \mathrm{DSPACE}(n^c) \end{aligned}$$

Relation between time and space

- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, if S is space constructible.

- Definition.

$$\begin{aligned} \mathrm{L} &= \mathrm{DSPACE}(\log n) \\ \mathrm{NL} &= \mathrm{NSPACE}(\log n) \\ \mathrm{PSPACE} &= \bigcup_{c>0} \mathrm{DSPACE}(n^c) \end{aligned}$$

Giving space at least log n gives a TM at least the power to remember the index of a cell.

Relation between time and space

- Obs. DTIME(S(n)) $\subsetneq$ DSPACE(S(n)) $\subseteq$ NSPACE(S(n)).
- Theorem. NSPACE(S(n)) $\subseteq$ DTIME($2^{O(S(n))}$), if S is space constructible.
- Caution. The *Hopcroft-Paul-Valiant* theorem does not imply P $\subsetneq$ PSPACE.
- Open. Is P $\neq$ PSPACE ?

Relation between time and space

- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.
- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, if S is space constructible.
- Theorem. $\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$

Follows from the above theorem

Relation between time and space

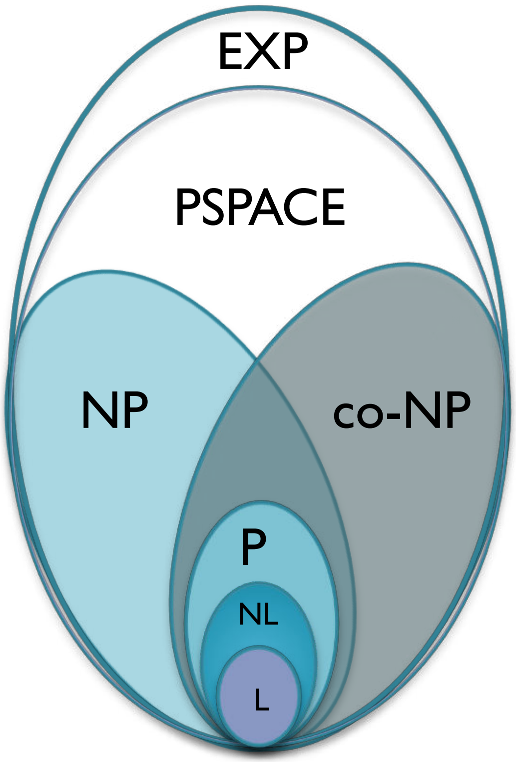
- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.
- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, if S is space constructible.
- Theorem. $\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$

Run through all possible choices of certificates of the verifier and **reuse** space.

Relation between time and space

- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, if S is space constructible.



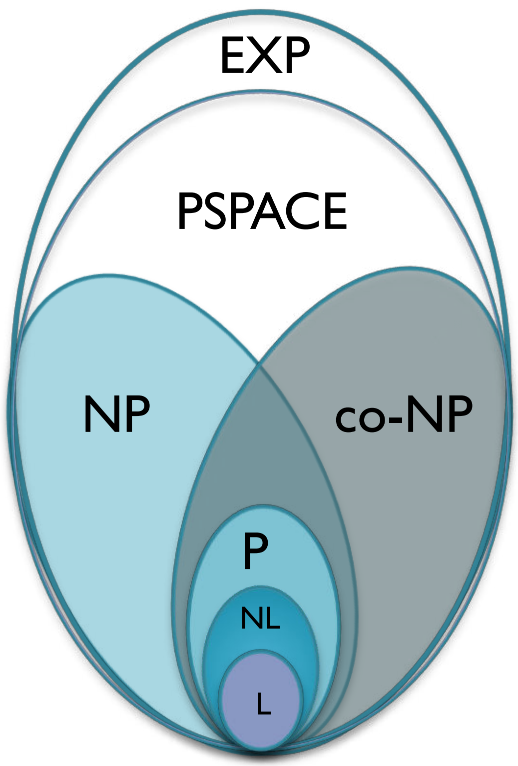
Relation between time and space

- Obs. DTIME(S(n)) $\subsetneq$ DSPACE(S(n)) $\subseteq$ NSPACE(S(n)).

- Theorem. NSPACE(S(n)) $\subseteq$ DTIME($2^{O(S(n))}$), if S is space constructible.

Homework: Integer addition and multiplication are in (functional) L.

Integer division is also in (functional) L. *(Chiu, Davida & Litow 2001)*



Configuration graph

- Definition. A *configuration* of a TM M on input x, at any particular step of its execution, consists of

  (a) the nonblank symbols of its work tapes,

  (b) the current state,

  (c) the current head positions.

It captures a 'snapshot' of M at any particular moment of execution.

Configuration graph

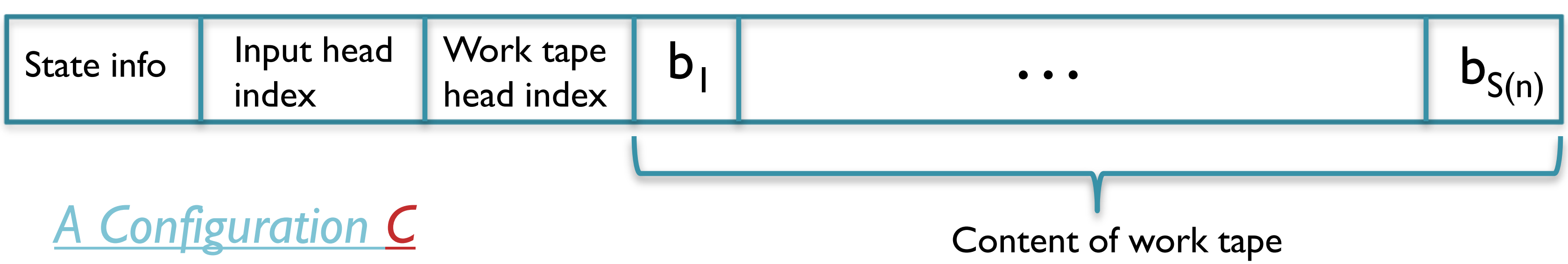
- Definition. A *configuration* of a TM M on input x, at any particular step of its execution, consists of

    (a) the nonblank symbols of its work tapes,

    (b) the current state,

    (c) the current head positions.

It captures a 'snapshot' of M at any particular moment of execution.



A Configuration C

Configuration graph

- Definition. A *configuration* of a TM M on input x, at any particular step of its execution, consists of
  - (a) the nonblank symbols of its work tapes,
  - (b) the current state,
  - (c) the current head positions.

It captures a 'snapshot' of M at any particular moment of execution.

State info	Input head index	Work tape head index	b_1	. . .	$b_{S(n)}$

Note: A configuration C can be represented using $O(S(n))$ bits if M uses $S(n) = \Omega(\log n)$ space on n-bit inputs.

Configuration graph

- Definition. A *configuration graph* of a TM M on input x, denoted $G_{M,x}$, is a directed graph whose nodes are all the possible configurations of M on input x. There's an edge from one configuration C_1 to another C_2, if C_2 can be reached from C_1 by an application of M's transition function(s).

Configuration graph

- Definition. A *configuration graph* of a TM M on input x, denoted $G_{M,x}$, is a directed graph whose nodes are all the possible configurations of M on input x. There's an edge from one configuration C_1 to another C_2, if C_2 can be reached from C_1 by an application of M's transition function(s).

- Number of nodes in $G_{M,x}$ = $2^{O(S(n))}$, if M uses $S(n)$ space on n-bit inputs

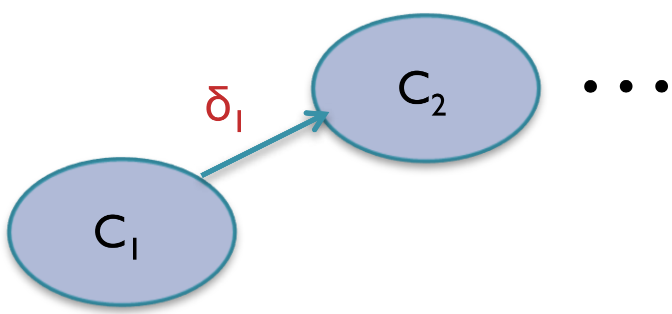
Configuration graph

- Definition. A *configuration graph* of a TM M on input x, denoted $G_{M,x}$, is a directed graph whose nodes are all the possible configurations of M on input x. There's an edge from one configuration C_1 to another C_2, if C_2 can be reached from C_1 by an application of M's transition function(s).

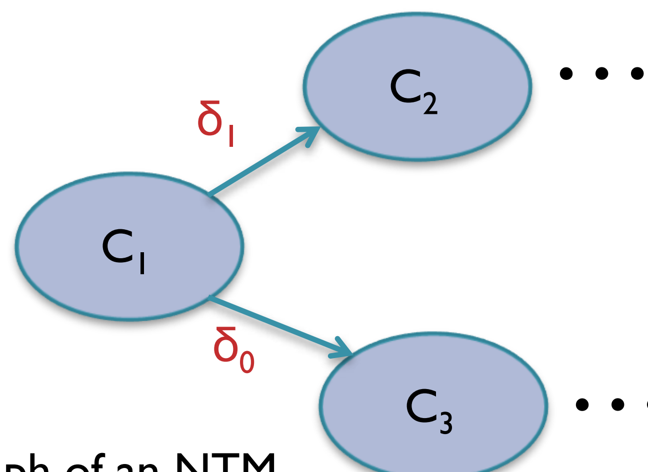
- If M is a DTM then every node C in $G_{M,x}$ has at most one outgoing edge. If M is an NTM then every node C in $G_{M,x}$ has at most two outgoing edges.

Configuration graph

- Definition. A *configuration graph* of a TM M on input x, denoted $G_{M,x}$, is a directed graph whose nodes are all the possible configurations of M on input x. There's an edge from one configuration C_1 to another C_2, if C_2 can be reached from C_1 by an application of M's transition function(s).



Conf. graph of a DTM



Conf. graph of an NTM

Configuration graph

- Definition. A *configuration graph* of a TM M on input x, denoted $G_{M,x}$, is a directed graph whose nodes are all the possible configurations of M on input x. There's an edge from one configuration C_1 to another C_2, if C_2 can be reached from C_1 by an application of M's transition function(s).

- By erasing the contents of the work tape at the end, bringing the head at the beginning, and having a q_{accept} state, we can assume that there's a unique C_{accept} configuration. Configuration C_{start} is well defined.

Configuration graph

- Definition. A *configuration graph* of a TM M on input x, denoted $G_{M,x}$, is a directed graph whose nodes are all the possible configurations of M on input x. There's an edge from one configuration C_1 to another C_2, if C_2 can be reached from C_1 by an application of M's transition function(s).

- M accepts x if and only if there's a path from C_{start} to C_{accept} in $G_{M,x}$.

Relation between time and space

- Obs. $\mathrm{DTIME}(S(n)) \subsetneq \mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$, if S is space constructible.

- Proof. Let $L \in \mathrm{NSPACE}(S(n))$ and M be an NTM deciding L using $O(S(n))$ space on length n inputs.
- On input x, compute the configuration graph $G_{M,x}$ of M and check if there's a **path** from C_{start} to C_{accept} . Running time is $2^{O(S(n))}$.

Natural problems?

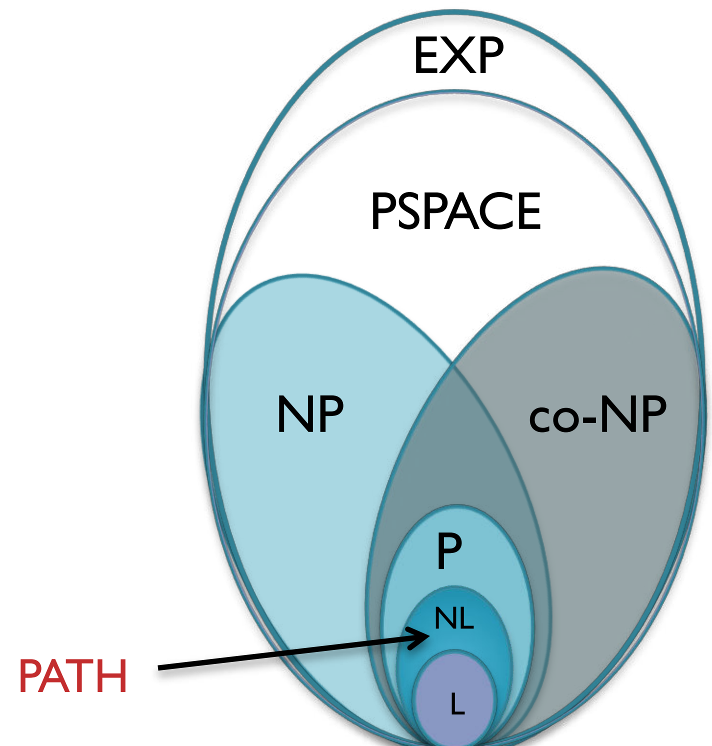
- Definition. $$\begin{aligned} L &= \mathrm{DSPACE}(\log n) \\ NL &= \mathrm{NSPACE}(\log n) \\ PSPACE &= \bigcup_{c>0} \mathrm{DSPACE}(n^c) \end{aligned}$$

- Theorem. $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXP$.

- Are there natural problems in L, NL and PSPACE ?

PATH: A canonical problem in NL

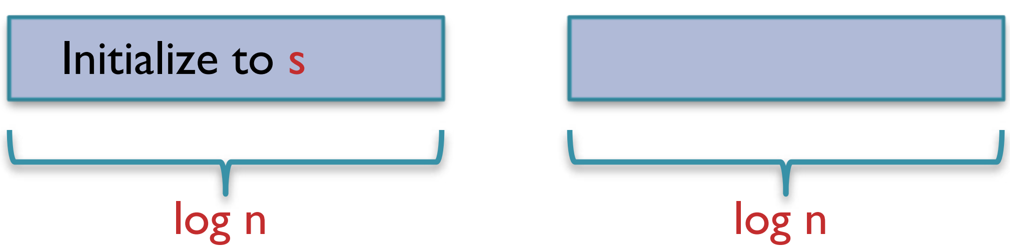
- PATH = {(G,s,t) : G is a directed graph having a path from s to t}.
- Obs. PATH is in NL.



PATH: A canonical problem in NL

- PATH = {(G,s,t) : G is a directed graph having a path from s to t}.
- Obs. PATH is in NL.

- Proof. Count the no. of vertices in G, let it be n. Set aside two memory locations of log n bits each. Initialize a counter, say Count = m < n.



Count = m

PATH: A canonical problem in NL

- PATH = {(G,s,t) : G is a directed graph having a path from s to t}.
- Obs. PATH is in NL.

- Proof. Count the no. of vertices in G, let it be n. Set aside two memory locations of log n bits each. Initialize a counter, say Count = m < n.

Initialize to s

Guess a vertex v_1

Count = m

If there's a edge from s to v_1, decrease count by 1.
Else o/p 0 and stop.

PATH: A canonical problem in NL

- PATH = {(G,s,t) : G is a directed graph having a path from s to t}.
- Obs. PATH is in NL.

- Proof. Count the no. of vertices in G, let it be n. Set aside two memory locations of log n bits each. Initialize a counter, say Count = m < n.

Set to v_1

Guess a vertex v_2

Count = m-1

If there's a edge from v_1 to v_2, decrease count by 1. Else o/p 0 and stop.

PATH: A canonical problem in NL

- PATH = {(G,s,t) : G is a directed graph having a path from s to t}.
- Obs. PATH is in NL.

- Proof. Count the no. of vertices in G, let it be n. Set aside two memory locations of log n bits each. Initialize a counter, say Count = m < n.

Set to v_2

Guess a vertex v_3

Count = m-2

If there's a edge from v_2 to v_3, decrease count by 1. Else o/p 0 and stop.

…and so on.

PATH: A canonical problem in NL

- PATH = {(G,s,t) : G is a directed graph having a path from s to t}.
- Obs. PATH is in NL.

- Proof. Count the no. of vertices in G, let it be n. Set aside two memory locations of log n bits each. Initialize a counter, say Count = m < n.

Set to v_{m-1}

Set to t

Count = 1

If there's a edge from v_{m-1} to t, o/p 1 and stop.
Else o/p 0 and stop.

PATH: A canonical problem in NL

- PATH = {(G,s,t) : G is a directed graph having a path from s to t}.
- Obs. PATH is in NL.

- Proof. Count the no. of vertices in G, let it be n. Set aside two memory locations of log n bits each. Initialize a counter, say Count = m < n.

Set to v_{m-1}

Set to t

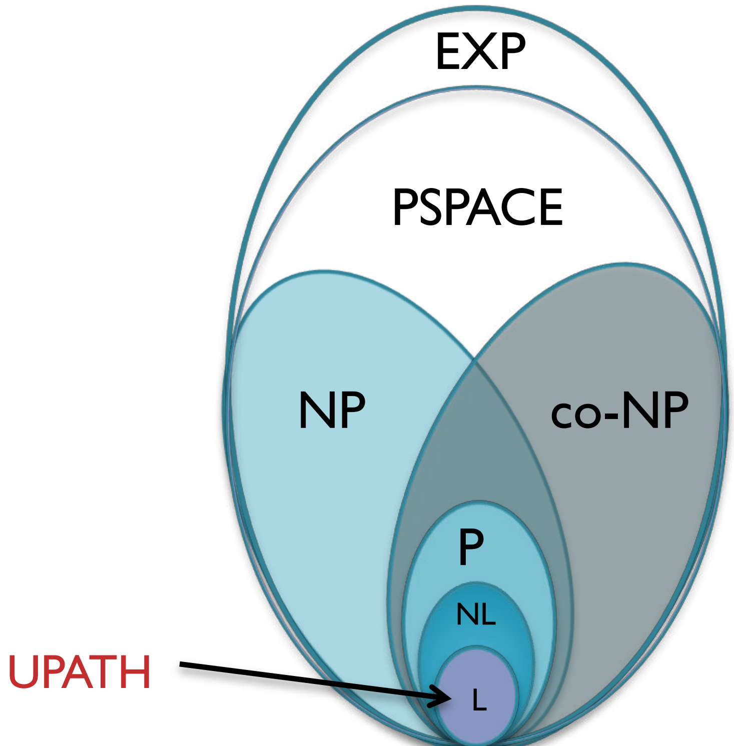
Count = 1

If there's a edge from v_{m-1}
to t, o/p 1 and stop.
Else o/p 0 and stop.

Space complexity = O(log n)

UPATH: A problem in L

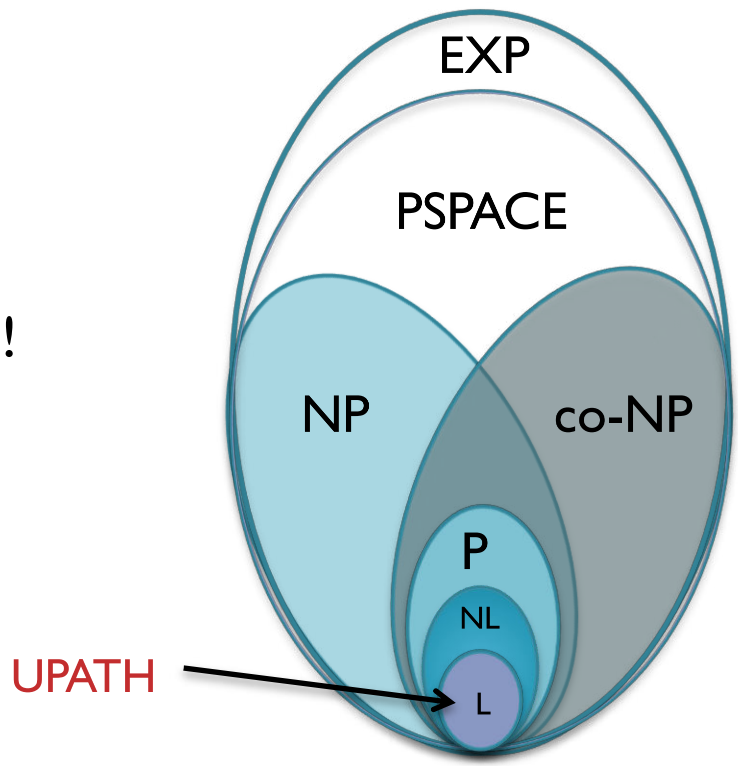
- UPATH = {(G,s,t) : G is an undirected graph having a path from s to t}.
- Theorem *(Reingold 2005).* UPATH is in L.



UPATH: A problem in L

- UPATH = {(G,s,t) : G is an undirected graph having a path from s to t}.
- Theorem *(Reingold 2005).* UPATH is in L.

Is PATH in L ?
If yes, then L = NL !
(will prove later)



Space Hierarchy Theorem

- Theorem. *(Stearns, Hartmanis & Lewis 1965)* If f and g are space-constructible functions and $f(n) = o(g(n))$, then $\mathrm{SPACE}(f(n)) \subsetneq \mathrm{SPACE}(g(n))$.

- Proof. Homework.

- Theorem. $\mathrm{L} \subsetneq \mathrm{PSPACE}$.

PSPACE = NPSPACE

Savitch's theorem

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DSPACE}(S(n)^2)$, where $S(n)$ is space constructible. (So, PSPACE = NPSPACE)

- Proof. Let $L \in \mathrm{NSPACE}(S(n))$, and M be an NTM requiring $O(S(n))$ space to decide L. We'll show that there's a TM N requiring $O(S(n)^2)$ space to decide L.

Savitch's theorem

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DSPACE}(S(n)^2)$, where $S(n)$ is space constructible. (So, PSPACE = NPSPACE)

- Proof. Let $L \in \mathrm{NSPACE}(S(n))$, and M be an NTM requiring $O(S(n))$ space to decide L. We'll show that there's a TM N requiring $O(S(n)^2)$ space to decide L.

- On input x, N checks if there's a path from C_{start} to C_{accept} in $G_{M,x}$ as follows: Let $|x| = n$.

Savitch's theorem

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DSPACE}(S(n)^2)$, where $S(n)$ is space constructible. (So, PSPACE = NPSPACE)

- Proof. (contd.) N computes $m = O(S(n))$, the no. of bits required to represent a configuration of M. It also finds out C_{start} and C_{accept}. Then N checks if there's a path from C_{start} to C_{accept} of length at most 2^m in $G_{M,x}$ recursively using the following procedure.

- $\mathrm{REACH}(C_1, C_2, i)$: returns 1 if there's a path from C_1 to C_2 of length at most 2^i in $G_{M,x}$; 0 otherwise.

Savitch's theorem

- Theorem. $NSPACE(S(n)) \subseteq DSPACE(S(n)^2)$, where $S(n)$ is space constructible. (So, PSPACE = NPSPACE)

- Proof. (contd.) $\overbrace{N \text{ computes } m = O(S(n))}^{\text{Space constructibility of } S(n) \text{ used here}}$, the no. of bits required to represent a configuration of M. It also finds out C_{start} and C_{accept}. Then N checks if there's a path from C_{start} to C_{accept} of length at most 2^m in $G_{M,x}$ recursively using the following procedure.

- $REACH(C_1, C_2, i)$: returns 1 if there's a path from C_1 to C_2 of length at most 2^i in $G_{M,x}$; 0 otherwise.

Savitch's theorem

- Theorem. NSPACE(S(n)) $\subseteq$ DSPACE(S(n)^2), where S(n) is space constructible. (So, PSPACE = NPSPACE)

- Proof.
- REACH(C_1, C_2, i) {

  If i = 0 check if C_1 and C_2 are adjacent.

  Else, <u>for every</u> configurations C,

  a_1 = REACH(C_1, C, i-1)

  a_2 = REACH(C, C_2, i-1)

  if a_1=1 & a_2=1, return 1. Else return 0.

}

Savitch's theorem

- Theorem. NSPACE(S(n)) $\subseteq$ DSPACE(S(n)^2), where S(n) is space constructible. (So, PSPACE = NPSPACE)

- Proof.
- REACH(C_1, C_2, i) {

  If i = 0 check if C_1 and C_2 are adjacent.

  Else, <u>for every</u> configurations C,

  a_1 = REACH(C_1, C, i-1)

  a_2 = REACH(C, C_2, i-1)

  if a_1=1 & a_2=1, return 1. Else return 0.

}

Require O(S(n)) space

Savitch's theorem

- Theorem. NSPACE(S(n)) $\subseteq$ DSPACE(S(n)^2), where S(n) is space constructible. (So, PSPACE = NPSPACE)

- Proof.
- REACH(C_1, C_2, i) {

  If i = 0 check if C_1 and C_2 are adjacent.

  Else, <u>for every</u> configurations C,

  $$\left.\begin{array}{l} a_1 = \text{REACH}(C_1, C, i-1) \\ a_2 = \text{REACH}(C, C_2, i-1) \end{array}\right\} \text{Reuse space}$$

  if a_1=1 & a_2=1, return 1. Else return 0.

}

Savitch's theorem

- Theorem. $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DSPACE}(S(n)^2)$, where $S(n)$ is space constructible. (So, PSPACE = NPSPACE)

- Proof.

$$\mathrm{Space}(i) = \mathrm{Space}(i-1) + O(S(n))$$

- Space complexity: $O(S(n)^2)$

Savitch's theorem

- Theorem. NSPACE(S(n)) $\subseteq$ DSPACE(S(n)^2), where S(n) is space constructible. (So, PSPACE = NPSPACE)

- Proof.

$$\text{Space}(i) = \text{Space}(i-1) + O(S(n))$$

- Space complexity: $O(S(n)^2)$

$$\text{Time}(i) = 2^m.2.\text{Time}(i-1) + O(S(n))$$

- Time complexity: $2^{O(S(n)^2)}$

Savitch's theorem

- Theorem. $\text{NSPACE}(S(n)) \subseteq \text{DSPACE}(S(n)^2)$, where $S(n)$ is space constructible. (So, PSPACE = NPSPACE)

- Proof.

$$\text{Space}(i) = \text{Space}(i\text{-}1) + O(S(n))$$

- Space complexity: $O(S(n)^2)$

$$\text{Time}(i) = 2^m.2.\text{Time}(i\text{-}1) + O(S(n))$$

- Time complexity: $2^{O(S(n)^2)}$

Recall, $\text{NSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))})$. There's an algorithm with time complexity $2^{O(S(n))}$, but higher space requirement.

PSPACE-completeness

PSPACE-completeness

- Recall, to define completeness of a complexity class, we need an appropriate notion of a ***reduction***.
- What kind of reductions will be suitable is guided by ***a complexity question***, like a comparison between the complexity class under consideration & another class.
- Is P = PSPACE ?

PSPACE-completeness

- Recall, to define completeness of a complexity class, we need an appropriate notion of a *reduction*.
- What kind of reductions will be suitable is guided by *a complexity question*, like a comparison between the complexity class under consideration & another class.
- Is P = PSPACE ? …use poly-time Karp reduction!

- Definition. A language L' is *PSPACE-hard* if for every L in PSPACE, $L \leq_p L'$. Further, if L' is in PSPACE then L' is *PSPACE-complete*.

A PSPACE-complete problem

- Recall, to define completeness of a complexity class, we need an appropriate notion of a *reduction*.
- What kind of reductions will be suitable is guided by *a complexity question*, like a comparison between the complexity class under consideration & another class.
- Is P = PSPACE ? …use poly-time Karp reduction!

- Example. L' = {(M,w,1^m) : M accepts w using m space}

Natural PSPACE-complete problem

- Definition. A *quantified Boolean formula (QBF)* is a formula of the form

$$Q_1x_1\ Q_2x_2 \ldots Q_nx_n\ \ \underbrace{\phi(x_1, x_2, \ldots, x_n)}_{\text{Just a formula on Boolean variables}}$$

Quantifiers $\exists$ or $\forall$

- A QBF is either ***true*** or ***false*** as all variables are quantified. This is unlike a formula we've seen before where variables were unquantified/free.

Natural PSPACE-complete problem

- Example. $\exists x_1 \, \exists x_2 \ldots \exists x_n \;\; \phi(x_1, x_2, \ldots, x_n)$

- The above QBF is true iff ϕ is satisfiable.

- We could have defined SAT as

  $$\text{SAT} = \{\exists \mathbf{x}\, \phi(\mathbf{x}) : \phi \text{ is a CNF and } \exists \mathbf{x}\, \phi(\mathbf{x}) \text{ is true}\}$$

  instead of

  $$\text{SAT} = \{\phi(\mathbf{x}) : \phi \text{ is a CNF and } \phi \text{ is satisfiable}\}$$

Natural PSPACE-complete problem

- Definition. A *quantified Boolean formula (QBF)* is a formula of the form

$$Q_1 x_1 \; Q_2 x_2 \; \dots \; Q_n x_n \;\; \underbrace{\phi(x_1, x_2, \dots, x_n)}_{\text{Just a formula on Boolean variables}}$$

Quantifiers $\exists$ or $\forall$

- Homework: By using auxiliary variables (as in the proof of Cook-Levin) and introducing some more $\exists$ quantifiers at the end, we can assume w.l.o.g. that ϕ is a 3CNF.

Natural PSPACE-complete problem

- Definition. TQBF is the set of ***<u>true</u>*** quantified Boolean formulas.

- Theorem. TQBF is PSPACE-complete.