Computational Complexity Theory

Lecture 11: Space complexity classes (contd.); Savitch's theorem

Department of Computer Science,
Indian Institute of Science

Recap: 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 DSPACE(S(n)) if there's a TM M that decides L using O(S(n)) work space on inputs of length n.

Recap: 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 $S: \mathbb{N} \to \mathbb{N}$ 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.

Recap: 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.

Recap: 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.

$$L = \mathrm{DSPACE}(\log n)$$

$$NL = \mathrm{NSPACE}(\log n)$$

$$\mathrm{PSPACE} = \bigcup_{c>0} \mathrm{DSPACE}(n^c)$$

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

Recap: 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.

$$\mathrm{L} = \mathrm{DSPACE}(\log n)$$

$$\mathrm{NL} = \mathrm{NSPACE}(\log n)$$

$$\mathrm{PSPACE} = \bigcup_{c > 0} \mathrm{DSPACE}(n^c)$$

Why did we not define NPSPACE?
We'll see that, unlike P and NP,
PSPACE = NPSPACE

Recap: 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 ?

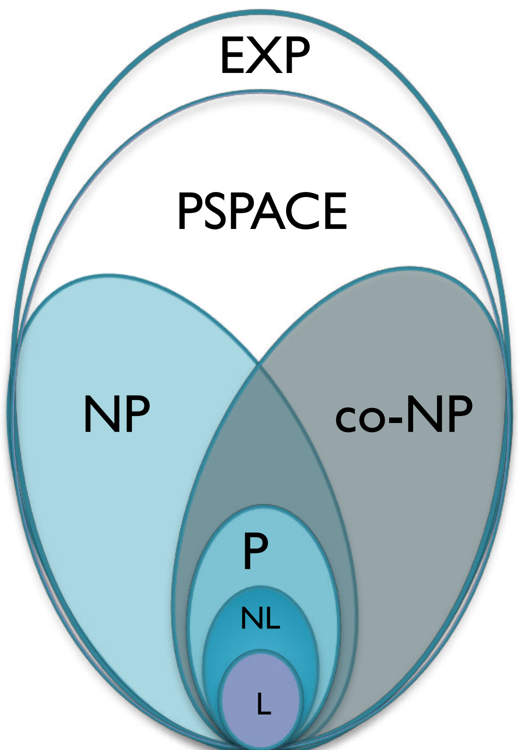
Recap: Time and space

- 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,

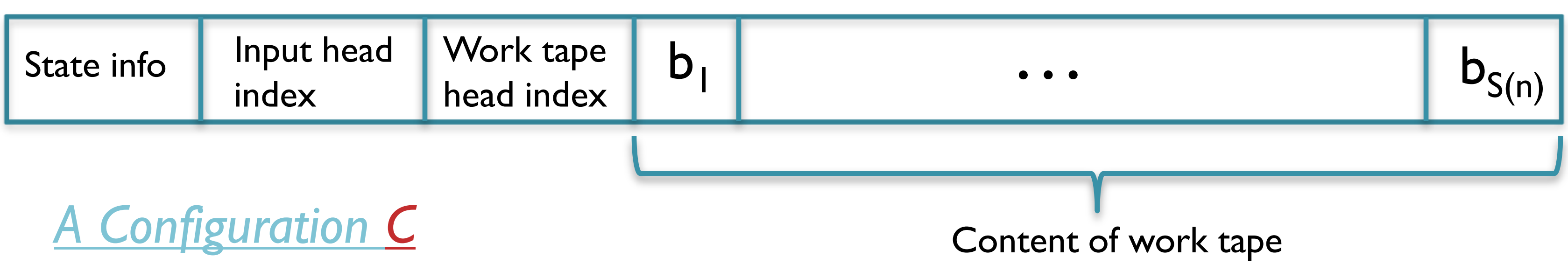
    (c) the current head positions.

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

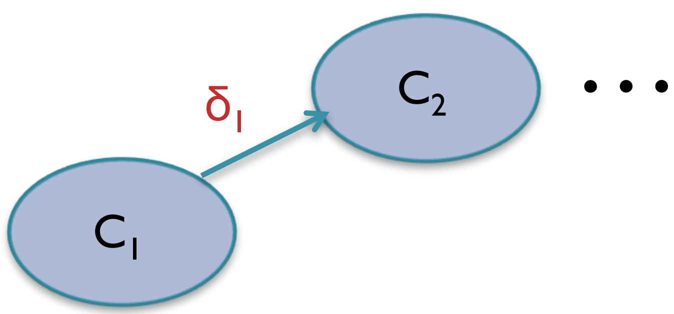
Configuration graph

- 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).

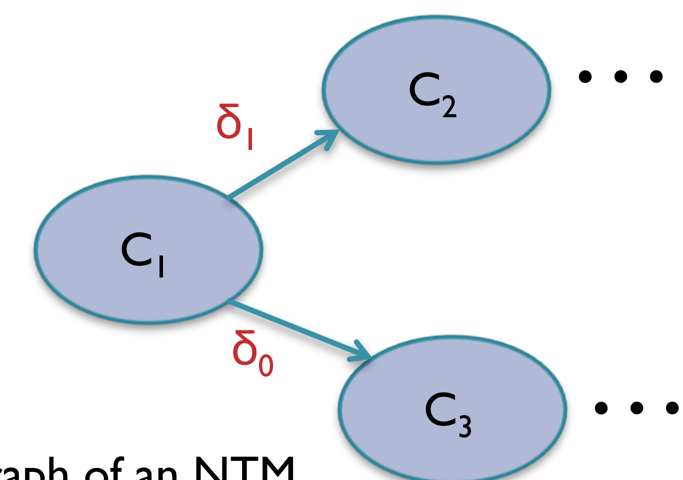
- If M is a DTM then every node C in $G_{M,x}$ has at most one outgoing edge. If M is an NTM then every node C in $G_{M,x}$ has at most two outgoing edges.

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. 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.

- Proof. Let L $\in$ 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. $\quad L = DSPACE(\log n)$

  $NL = NSPACE(\log n)$

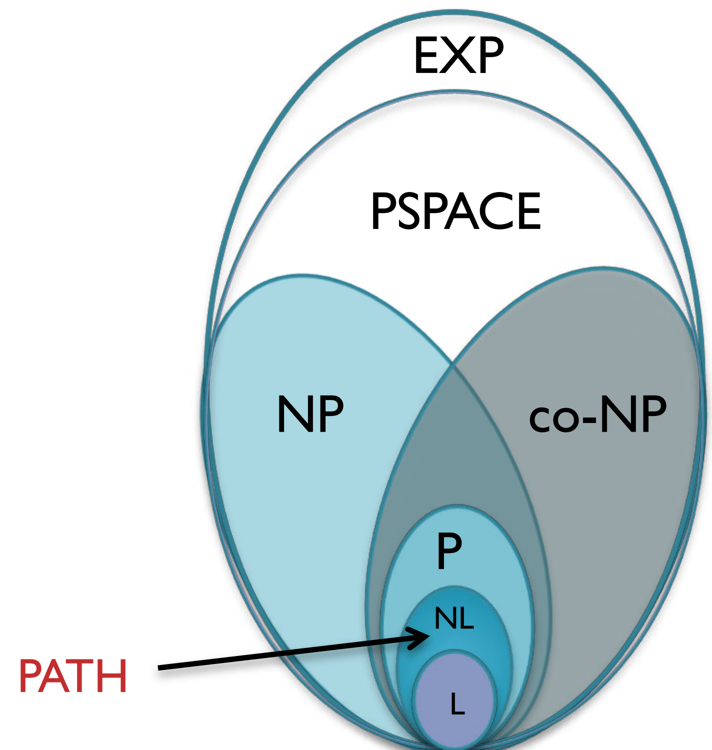
  $PSPACE = \bigcup_{c > 0} DSPACE(n^c)$

- 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

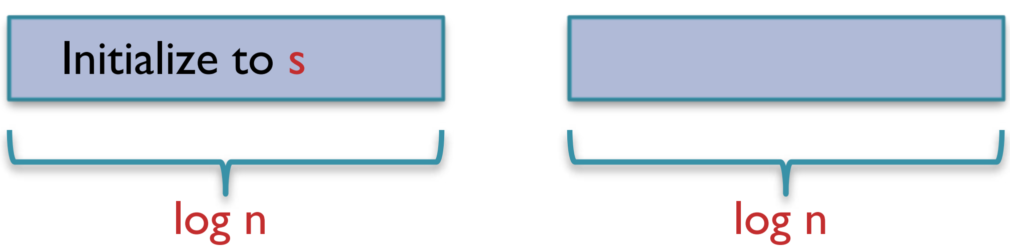
- PATH = {(G,s,t) : G is a directed graph having a path from s to t}.
- Obs. PATH is in NL.



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

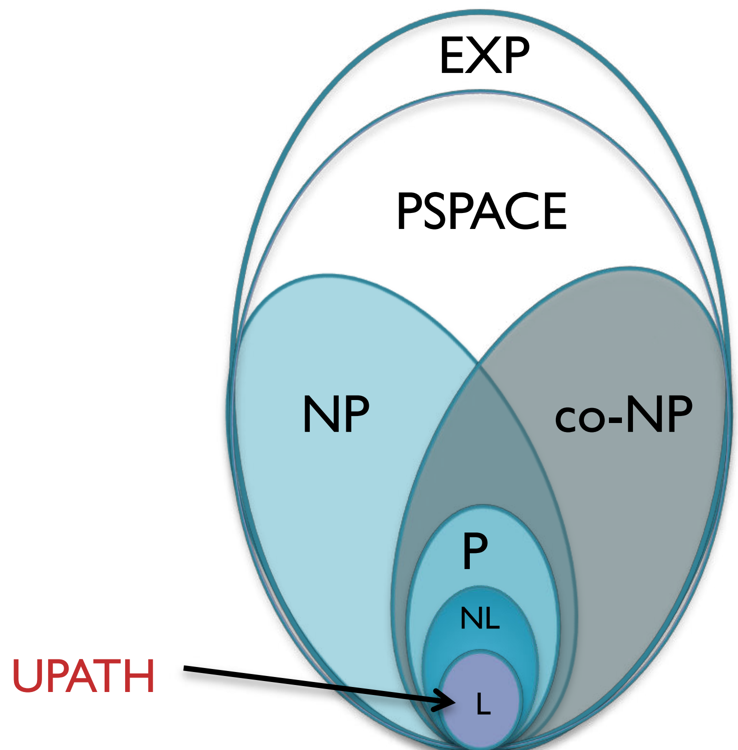
Count = 1

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.



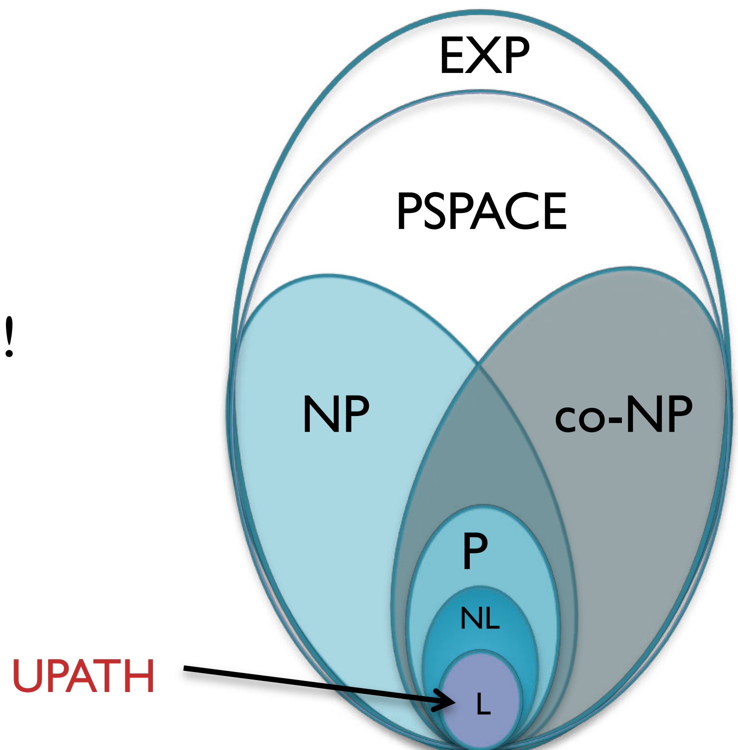
UPATH: A problem in L

- 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 $\text{SPACE}(f(n)) \subsetneq \text{SPACE}(g(n))$.

- Proof. Homework.

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

PSPACE = NPSPACE

Savitch's theorem

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

- Proof. Let L $\in$ 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. NSPACE(S(n)) ⊆ DSPACE($S(n)^2$), where S(n) is space constructible. (So, PSPACE = NPSPACE)

- Proof. Let L ∈ 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. NSPACE(S(n)) $\subseteq$ 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.

- 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)

Space constructibility of S(n) used here

- 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.

- 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, for every 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)) ⊆ DSPACE(S(n)^2), where S(n) is space constructible. (So, PSPACE = NPSPACE)

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

  Require O(S(n)) space

  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)) ⊆ 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)

    } Reuse space

    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.

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

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

- Proof.

$$Space(i) = Space(i-1) + O(S(n))$$

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

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

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

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