Reading

Sipser: Chapter 8; or

Hopcroft et al.: 11.2, 11.3

Space Complexity

Space: a measure of the working storage (memory) used by Turing machines.

A simple definition: A Turing machine M operates within space bound $f(n)$ if for any input x of length n, M reads/writes at most $f(n)$ tape squares.

- This definition treats the input as part of the working storage. In other words, $f(n) \geq n$.
- E.g., To check whether a binary string has odd parity, we don't need any working storage, yet any Turing machine for this problem operates in space $\geq n$.

A better definition

We consider Turing machines with at least 2 tapes.

- first tape: contains the input; read only;
- other tapes: read/write working tapes.

We measure the space (working storage) used by a Turing machine with respect to the read/write tapes only.

Definition: A Turing machine M is said to operate within space bound $f(n)$ if for any input x of length n, M reads/writes a total of at most $f(n)$ tape squares on non-input tapes.

Space Complexity Classes

Let $f(n)$ be an integer function.

- SPACE ($f(n)$) = { L | L is a language decided by a deterministic Turing machine operating in space $O(f(n))$ }.

- NSPACE ($f(n)$) = { L | L is a language decided by a nondeterministic Turing machine operating in space $O(f(n))$ }.

Example 1

Recall that SAT is the set of all satisfiable Boolean formulas.

SAT $\in$ NP

SAT $\in$ TIME ($2^{O(n)}$)   (We believe that SAT $\notin$ P)

What is the (deterministic) space complexity of SAT ?

SAT $\in$ SPACE (?)

Example 1

SAT $\in$ TIME ($2^{O(n)}$)   (We believe that SAT $\notin$ P)

What is the (deterministic) space complexity of SAT?

SAT $\in$ SPACE (n)

Input: a Boolean Formula F

For every truth assignment A to the variables of F,
evaluate F w.r.t. A;
if "true" then accept.

Reject.

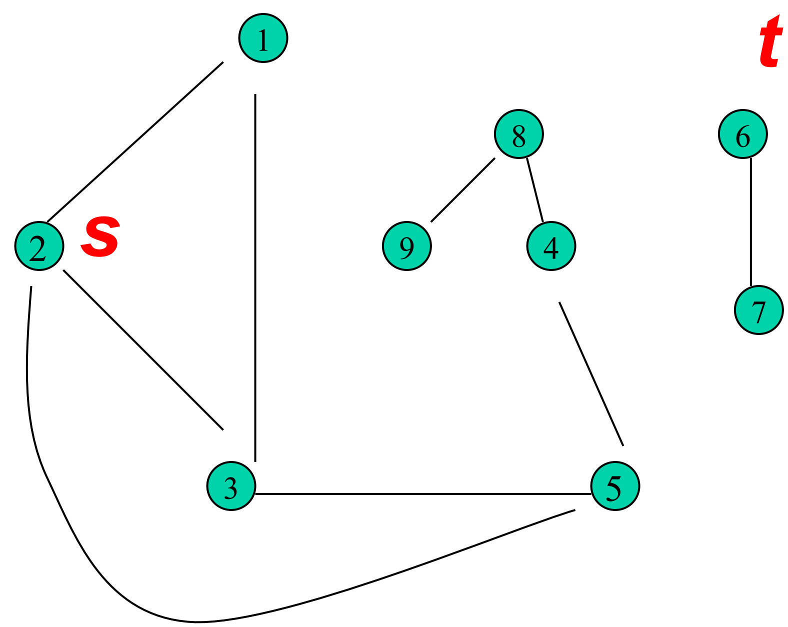
NB. Space for storing A and evaluation: O(n)

Example 2

Graph connectivity problem.

Given (G,s,t), an undirected graph G with n vertices (and at most n^2 edges) and two designated vertices s and t, determine whether there is a path from s to t.

This problem is in SPACE (n).



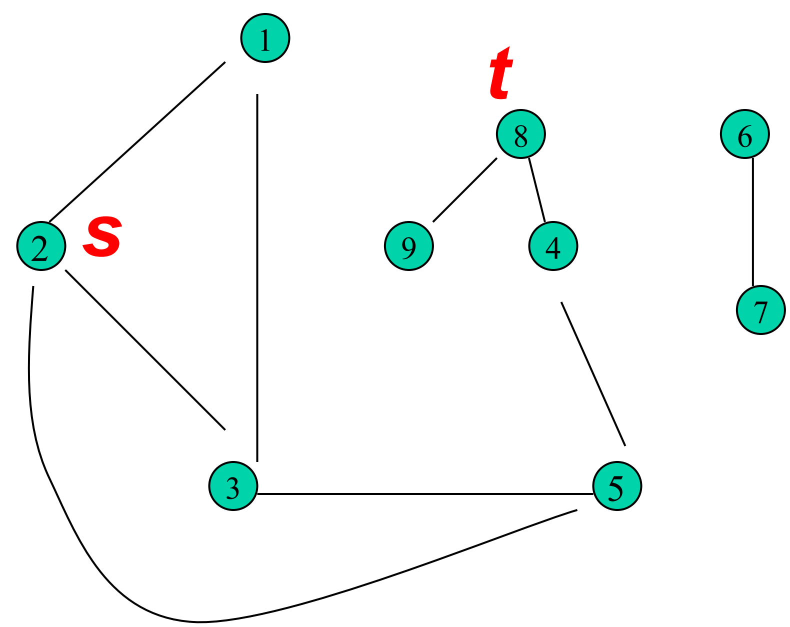
Example 3: nondeterministic space

The graph connectivity problem is in NSPACE (log n).

```
u = s;  count = 0;

while (u ≠ t  and count ≤ n)
        "guess" the next vertex v;
        if (u, v) is an edge,
            then u = v;
            else report "no"

if u = t, report "yes" else report "no"
```



NB. u, v, count occupies O(log n) bits.

Remarks on graph connectivity

Undirected graphs:

SPACE (n)

→ SPACE ($\log^2 n$) today's lecture

→ SPACE ($\log^{3/2} n$) [FOCS 1989]

→ SPACE ($\log^{4/3} n$) [STOC 96; JACM 2000]

→ SPACE ($\log n \log\log n$) [STOC 05]

→ SPACE ($\log n$) [STOC 05]

Directed graphs:

SPACE (n) → SPACE ($\log^2 n$)

Trivial facts on Time versus Space

True or False:

$TIME(f(n)) \subseteq NTIME(f(n))$

$SPACE(f(n)) \subseteq NSPACE(f(n))$

$TIME(f(n)) \subseteq SPACE(f(n))$

Roughly speaking, using time t, a Turing machine can read/write at most t squares.

$NTIME(f(n)) \subseteq NSPACE(f(n))$

$\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$

Theorem: $\mathrm{NTIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$

Theorem (Savitch): $\mathrm{NSAPCE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$, where $f(n) \geq \log n$.

Theorem. NTIME (f(n)) $\subseteq$ SPACE (f(n)).

Let L be any language in NTIME (f(n)) . By definition, L can be decided by an NTM T using O(f(n)) time.

c f(n), where c is some constant.

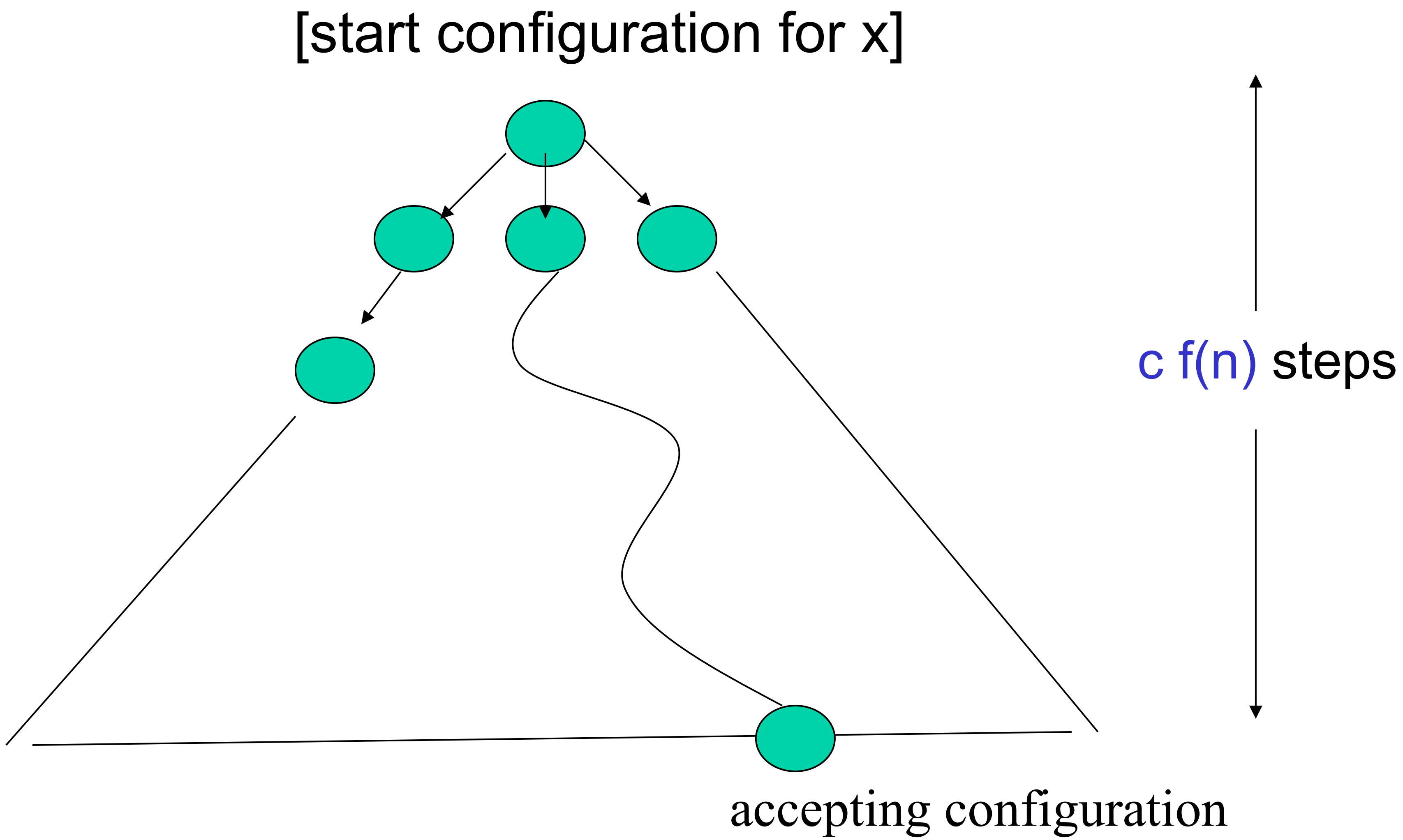
T is nondeterministic. In one step, T has more than one choice to move. Let d denote the maximum number of choices T can make in one step.

$d = \max \{ \ |\delta(q, a_1, a_2, \ldots, a_k)| \ \mid q \in Q$ and $a_i \in \Gamma \}$.

For any input x, the computation of T defines a tree.

If $x \in L$ and $|x|=n$, then there is a path from the start configuration to an accepting configuration within c f(n) steps.

Consider the computation tree of an input in x.



How can we represent such a path?

Answer 1: a sequence of $c\, f(n)$ configurations.

Note that each configuration occupies $O(f(n))$ space and the whole sequence uses $O(f^2(n))$ space.

Answer 2: start configuration + a sequence of move choices in each subsequent step.

Note that each choice can be represented by a number in the range $[1..d]$, occupying $O(1)$ space. The whole sequence uses $O(f(n))$ space.

Deterministic Searching of an accepting path of an NTM T

Let $n = |x|$ and $m = c\,f(n)$.

For **all** possible sequences $S = (a_1\ a_2\ \dots\ a_m)$ numbers, where each $a_i \in \{1, 2, \dots, d\}$.

- $i = 1$; C = the start configuration of T for input x.
- Repeat m times

  C' = Based on the transition function δ of T, determine the next configuration from C if T follows the a_i-th choice.

  Accept if C is an accepting configuration of T;

  $C = C'$; $i=i+1$;
- Reject

Analysis of space

Note that the whole procedure can be implemented on a deterministic Turing machine **M**.

At any time, we need to represent the following variables:

S: $O(f(n))$ space;

C,C': $O(f(n))$ space;

i: $\log c\, f(n)$ space

Thus, **M** can accept/reject an input x using $O(f(n))$ space.

In conclusion, for any NTM T using $O(f(n))$ time, we can construct a TM **M** using $O(f(n))$ space deciding the same language.

TIME (f(n)) $\subseteq$ NTIME (f(n)) $\subseteq$ SPACE (f(n)).

$\subseteq$

NSPACE (f(n))

$\subseteq$

SPACE ($f^2(n)$)

Theorem: NTIME (f(n)) $\subseteq$ SPACE (f(n)).

(Savitch's) Theorem: NSAPCE (f(n)) $\subseteq$ SPACE ($f^2(n)$),
where f(n) $\geq$ log n.

Savitch's Theorem

Theorem: NSAPCE (f(n)) $\subseteq$ SPACE ($f^2(n)$)

Consider any language L that can be decided by a k-tape NTM T using O(f(n)) space.

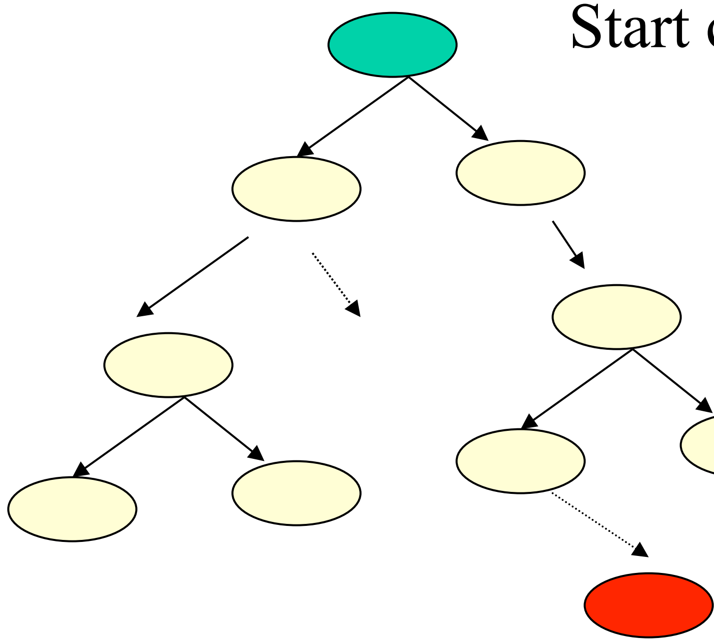
c f(n), where c is some constant.

We want to show that L in SPACE ($f^2(n)$). I.e., L can be decided by a (D)TM T' using O($f^2(n)$) space.

For any input x of length n, the computation of T defines a tree.

If $x \in L$, then this computation tree contains a path, say, **P**, ending with an accepting configuration.

If $x \notin L$, the **no** paths …. accepting configuration.



How many steps (configurations) are involved in in accepting "path" P ?

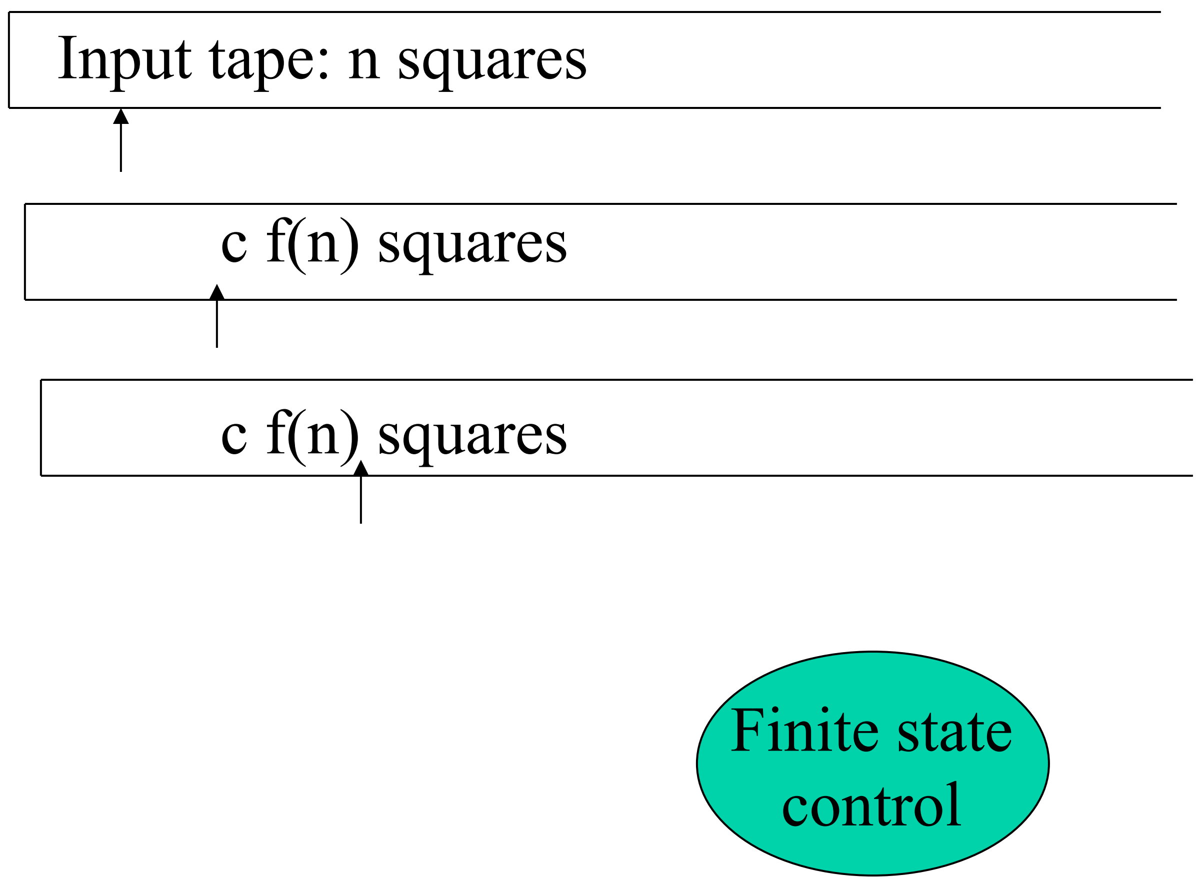
Note that we can choose **P** in such a way that no two configurations on P are the same.

To prove: at most $2^{O(f(n))}$ steps.

Idea. Bound the number of distinct configurations.

A configuration of T is characterised by 3 components:

- tape contents;
- tape head positions;
- current state.



T uses c f(n) space $\Rightarrow$ every configuration on P contains at most c f(n) non-blank squares on the working tapes.

The number of configurations on P is at most

$$(|\Gamma|^{c\,f(n)})^k \times (c\,f(n))^k \times n \times |Q|$$

# of possible tape symbols;
Γ is the tape alphabet

k working tape head positions

Input tape head position

which is $2^{d\,f(n)}$, where d is a big enough constant.

In other words, P takes at most $2^{d\,f(n)}$ steps.

Deciding L deterministically: using recursion

Given **T**, we construct a TM **T'** to decide L as follows:

For any input x (of length n),

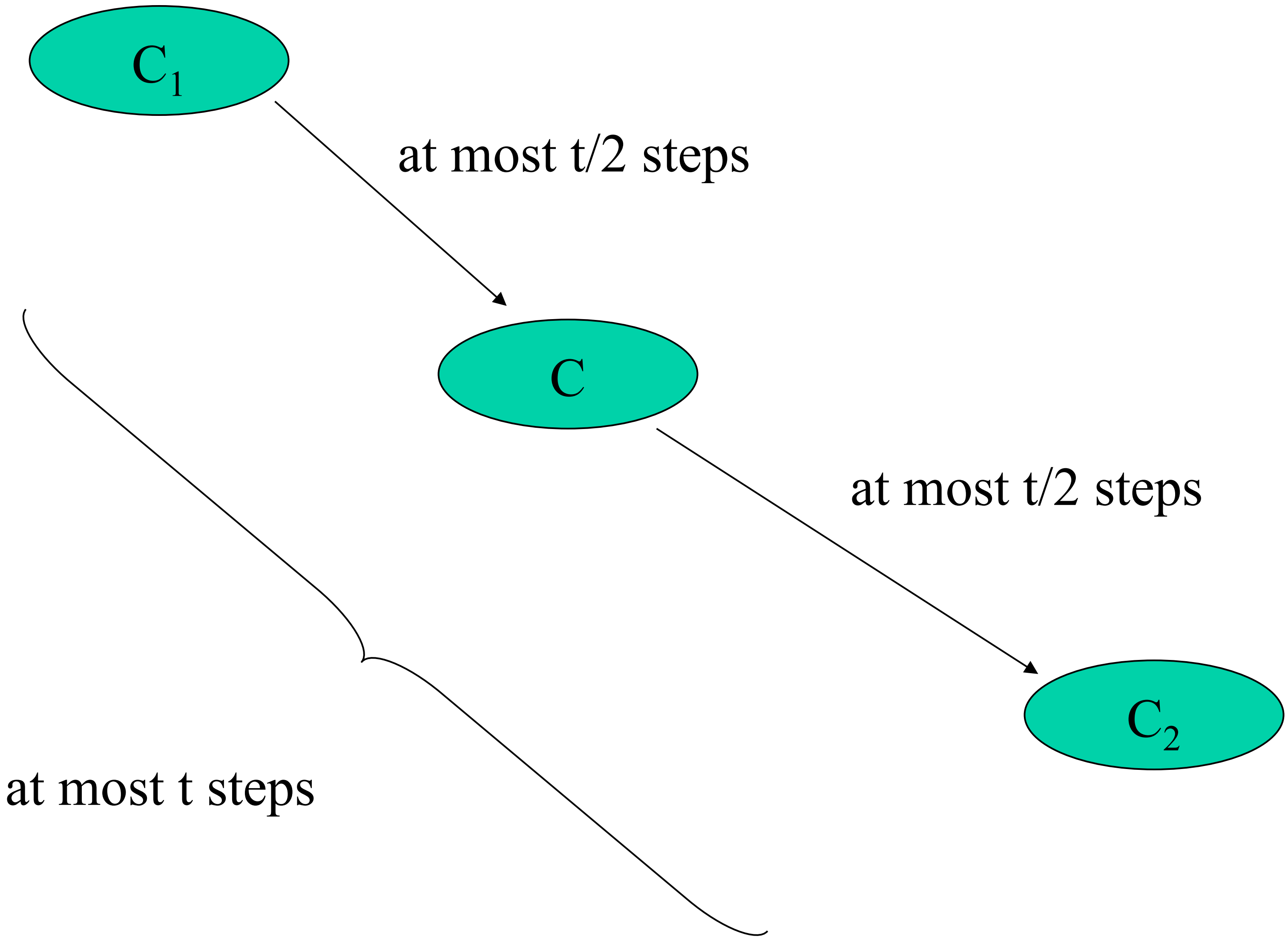
- Let $t_o = 2^{d\, f(n)}$.
- Let S be the start configuration of T for x.
- For <u>every possible</u> accepting configuration C_a for x,

  T' verifies whether T can move from S to C_a within t_o steps.

  [T' uses a recursive function CanReach (S, C_a, t_o).]

  If yes, **T'** accepts x immediately.

Can C_1 reach C_2 using t or fewer steps?



CanReach (C_1, C_2, t)

- If t = 1, if $C_1 = C_2$ or if T can move from C_1 to C_2 in one step, return "YES";
- If t > 1, then for every possible configuration C,

  if CanReach (C_1, C, t/2) and CanReach (C, C_2, t/2),

  then return "YES"
- Return "NO"

CanReach uses 4 local variables: C, C_1, C_2 occupies O(f(n)) space and t needs $\log(2^{d\, f(n)}) = O(f(n))$ space.

CanReach(C_1, C_2, t) is working deterministically.

It is a recursive procedure, the depth of recursion depends on t, precisely, at most log t.

In other words, the space required for the stack is at most log t $\times$ O(f(n)).

Thus, CanReach (S, C_a , t_o) uses $O(f^2(n))$ space.

Popular space complexity classes

PSPACE = $\cup_{i>0}$ SPACE (n^i)

What about **N**PSPACE?

It can be defined as $\cup_{i>0}$ NSPACE (n^i).

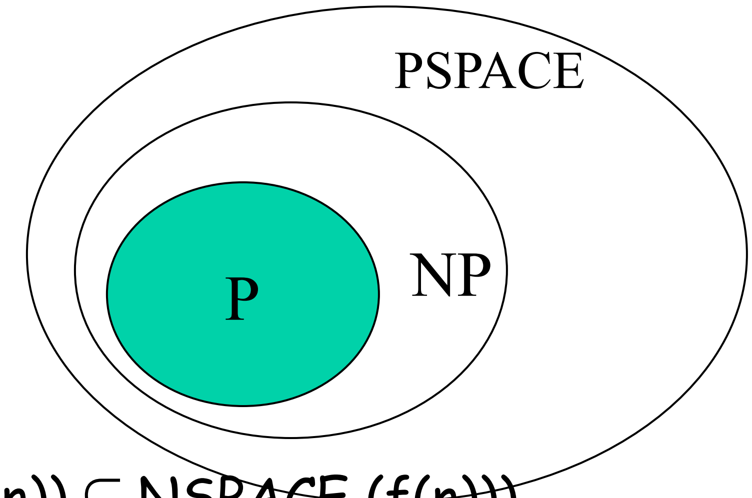
Lemma: PSPACE = NPSPACE.

- Obviously, PSPACE $\subseteq$ **N**PSPACE.
- By Savitch's theorem, for any $i > 0$,
  NSPACE (n^i) $\subseteq$ SPACE (n^{2i}) $\subseteq$ PSPACE.
- Therefore,
  NPSPACE = $\cup_{i>0}$ **N**SPACE (n^i) $\subseteq$ $\cup_{i>0}$ SPACE (n^i) = PSPACE.

P, NP, PSPACE

$P \subseteq$ PSPACE (since TIME $(f(n)) \subseteq$ SPACE $(f(n))$)

$NP \subseteq$ PSPACE



Two possible ways to prove it:

(a) $NP \subseteq$ NPSPACE (since NTIME $(f(n)) \subseteq$ NSPACE $(f(n))$)
= PSPACE

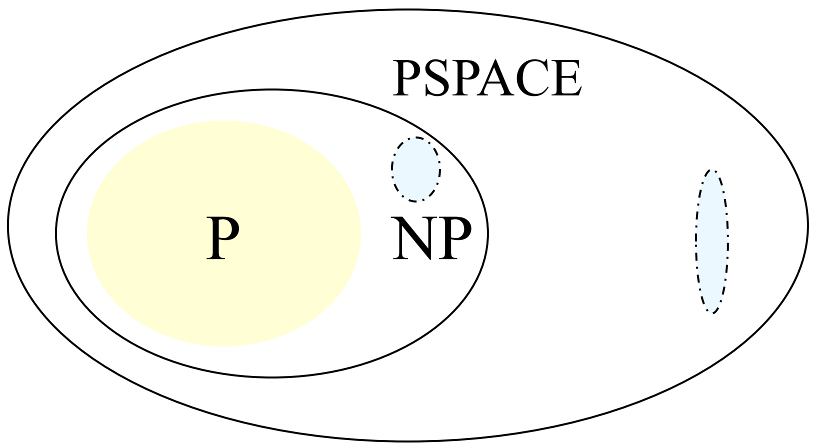
(b) NTIME $(n^i) \subseteq$ SPACE(n^i); thus,

$NP = \cup_{i>0}$ NTIME $(n^i) \subseteq \cup_{i>0}$ SPACE$(n^i) \subseteq$ PSPACE

PSPACE Complete

A language L is PSPACE-complete if

- L is in PSPACE, and
- every language L' in PSPACE is polynomial-time reducible to L.



Fact. Let L be a PSPACE-complete language. If L is in NP, then every language L' in PSPACE is also in NP, and PSPACE = NP.

Quantified Boolean formulas (QBF)

A QBF is a formula whose variables are quantified.

Two types of quantifiers:

- Universal quantifier (for all): $\forall x\ f(x)$ is true iff for every value of the variable x, f(x) is true.

  Example: $\forall x\, [\, x \vee \sim x]$

  $\forall x\ \forall y\, [\, x \wedge y\,]$

- Existential (there exists): $\exists x\ f(x)$ is true iff for some value of the variable x, f(x) is true (or equivalently, f(x) is satisfiable).

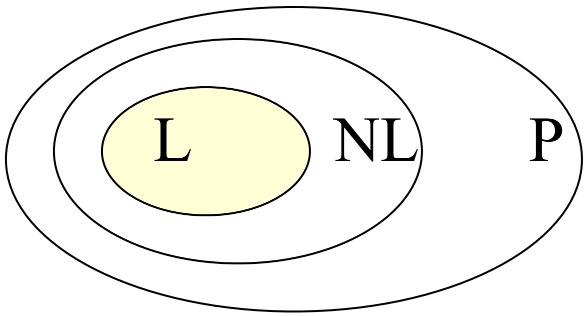
  Example: $\exists x\ \forall y\, [\, x \vee y\,]$

TBQF: Decide whether a given QBF is true.

Example: $\forall x\ \exists y\, [\, (x \vee y\,) \wedge (\sim x \vee \sim y)\,]$

$\exists y\ \forall x\, [\, (x \vee y\,) \wedge (\sim x \vee \sim y)\,]$

Other popular space complexity classes



L = SPACE (log n)

NL = NSPACE (log n)

It is believed that L is a proper subset of NL.

By Savitch's Theorem, NL $\subseteq$ SPACE($\log^2 n$).

Futhermore, NL $\subseteq$ P.

Consider the language PATH = { (G,s,t) | G is a directed graph containing a path from vertex s to vertex t }.

As shown before, PATH is in NL.

In fact, PATH is NL-complete. (Every language L in NL is log-space reducible to PATH).