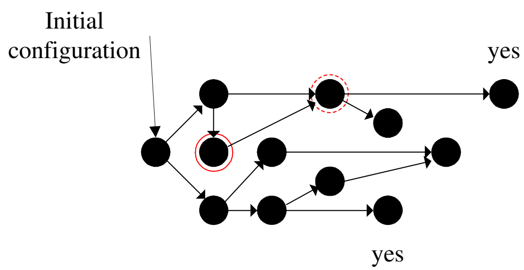
The Reachability Method

- The computation of a time-bounded TM can be represented by directional transitions between configurations.
- The reachability method constructs a directed graph with all the TM configurations as its nodes and edges connecting two nodes if one yields the other.
- The start node representing the initial configuration has zero in degree.
- When the TM is nondeterministic, a node may have an out degree greater than one.

Illustration of the Reachability Method



The reachability method may give the edges on the fly without explicitly storing the whole configuration graph.

Relations between Complexity Classes

Theorem 21 *Suppose $f(n)$ is proper. Then*

1. SPACE($f(n)$) $\subseteq$ NSPACE($f(n)$), TIME($f(n)$) $\subseteq$ NTIME($f(n)$).
2. NTIME($f(n)$) $\subseteq$ SPACE($f(n)$).
3. NSPACE($f(n)$) $\subseteq$ TIME($k^{\log n + f(n)}$).

- Proof of 2:
  - Explore the computation *tree* of the NTM for "yes."
  - Use the *depth-first* search as f is proper.

Proof of Theorem 21(2)

- (continued)
  - Specifically, generate a $f(n)$-bit sequence denoting the nondeterministic choices over $f(n)$ steps.
  - Simulate the NTM based on the choices.
  - Recycle the space and then repeat the above steps until a "yes" is encountered or the tree is exhausted.
  - Each path simulation consumes at most $O(f(n))$ space because it takes $O(f(n))$ time.
  - The total space is $O(f(n))$ as space is recycled.

Proof of Theorem 21(3)

- Let k-string NTM
$$M = (K, \Sigma, \Delta, s)$$
with input and output decide $L \in \text{NSPACE}(f(n))$.
- Use the reachability method on the configuration graph of M on input x of length n.
- A configuration is a $(2k+1)$-tuple
$$(q, w_1, u_1, w_2, u_2, \ldots, w_k, u_k).$$

Proof of Theorem 21(3) (continued)

- We only care about
$$(q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1}),$$
where i is an integer between 0 and n for the position of the first cursor.
- The number of configurations is therefore at most
$$|K| \times (n+1) \times |\Sigma|^{(2k-4)f(n)} = O(c_1^{\log n + f(n)}) \tag{2}$$
for some c_1, which depends on M.
- Add edges to the configuration graph based on M's transition function.

Proof of Theorem 21(3) (concluded)

- $x \in L \Leftrightarrow$ there is a path in the configuration graph from the initial configuration to a configuration of the form ("yes", $i, \ldots$) [there may be many of them].
- The problem is therefore that of REACHABILITY on a graph with $O(c_1^{\log n + f(n)})$ nodes.
- It is in TIME$(c^{\log n + f(n)})$ for some c because REACHABILITY is in TIME(n^k) for some k and
$$\left[c_1^{\log n + f(n)} \right]^k = (c_1^k)^{\log n + f(n)}.$$

The Grand Chain of Inclusions

$$\text{L} \subseteq \text{NL} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXP}.$$

- ~~It is known that PSPACE $\subsetneq$ EXP.~~
- By Corollary 20 (p. 177), we know L $\subsetneq$ PSPACE.
- The chain must break somewhere between L and PSPACE.
- It is suspected that all four inclusions are proper.
- But there are no proofs yet.[a]

[a]Carl Friedrich Gauss (1777–1855), "I could easily lay down a multitude of such propositions, which one could neither prove nor dispose of."

Nondeterministic Space and Deterministic Space

- By Theorem 5 (p. 92),

$$\text{NTIME}(f(n)) \subseteq \text{TIME}(c^{f(n)}),$$

an exponential gap.
- There is no proof that the exponential gap is inherent, however.
- How about NSPACE vs. SPACE?
- Surprisingly, the relation is only quadratic, a polynomial, by Savitch's theorem.

Savitch's Theorem

Theorem 22 (Savitch (1970))

$$\textsc{reachability} \in \text{SPACE}(\log^2 n).$$

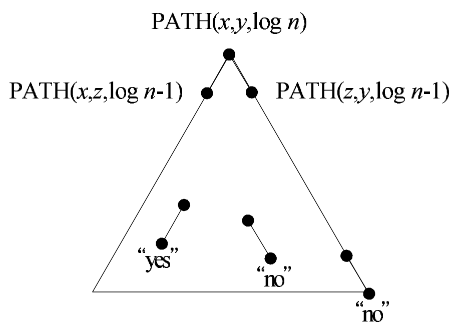
- Let G be a graph with n nodes.
- For $i \geq 0$, let

$$\text{PATH}(x, y, i)$$

mean there is a path from node x to node y of length at most 2^i.
- There is a path from x to y if and only if $\text{PATH}(x, y, \lceil \log n \rceil)$ holds.

The Proof (continued)

- For $i > 0$, $\text{PATH}(x, y, i)$ if and only if there exists a z such that $\text{PATH}(x, z, i-1)$ and $\text{PATH}(z, y, i-1)$.
- For $\text{PATH}(x, y, 0)$, check the input graph or if $x = y$.
- Compute $\text{PATH}(x, y, \lceil \log n \rceil)$ with a depth-first search on a graph with nodes (x, y, i)s (see next page).
- Like stacks in recursive calls, we keep only the current path of (x, y, i)s.
- The space requirement is proportional to the depth of the tree, $\lceil \log n \rceil$.



- Depth is $\lceil \log n \rceil$, and each node (x, y, i) needs space $O(\log n)$.
- The total space is $O(\log^2 n)$.

The Proof (concluded): Algorithm for PATH(x, y, i)

```
if i = 0 then
  if x = y or (x, y) ∈ G then
    return true;
  else
    return false;
  end if
else
  for z = 1, 2, ..., n do
    if PATH(x, z, i − 1) and PATH(z, y, i − 1) then
      return true;
    end if
  end for
  return false;
end if
```


The Relation between Nondeterministic Space and Deterministic Space Only Quadratic

Corollary 23 *Let $f(n) \geq \log n$ be proper. Then*

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)).$$

- Apply Savitch's theorem to the configuration graph of the NTM on the input.
- From p. 183, the configuration graph has $O(c^{f(n)})$ nodes; hence each node takes space $O(f(n))$.
- But if we supply the whole graph before applying Savitch's theorem, we get $O(c^{f(n)})$ space!

The Proof (continued)

- The way out is *not* to generate the graph at all.
- Instead, keep the graph implicit.
- We check for connectedness only when $i = 0$, by examining the input string.
- There, given configurations x and y, we go over the Turing machine's program to determine if there is an instruction that can turn x into y in one step.[a]

[a] Thanks to a lively class discussion on October 15, 2003.

The Proof (concluded)

- The z variable in the algorithm simply runs through all possible valid configurations.
- Each z has length $O(f(n))$ by Eq. (2) on p. 183.
- An alternative is to let $z = 0, 1, \ldots, O(c^{f(n)})$ and makes sure it is a valid configuration before using it in the recursive calls.[a]

[a] Thanks to a lively class discussion on October 13, 2004.

Implications of Savitch's Theorem

- PSPACE = NPSPACE.
- Nondeterminism is less powerful with respect to space.
- It may be very powerful with respect to time as it is not known if P = NP.

Nondeterministic Space Is Closed under Complement

- Closure under complement is trivially true for deterministic complexity classes (p. 170).
- It is known that[a]

$$\text{coNSPACE}(f(n)) = \text{NSPACE}(f(n)). \tag{3}$$

- So

$$\begin{aligned} \text{coNL} &= \text{NL}, \\ \text{coNPSPACE} &= \text{NPSPACE}. \end{aligned}$$

- But there are still no hints of coNP = NP.

[a] Szelepscényi (1987) and Immerman (1988).

Reductions and Completeness

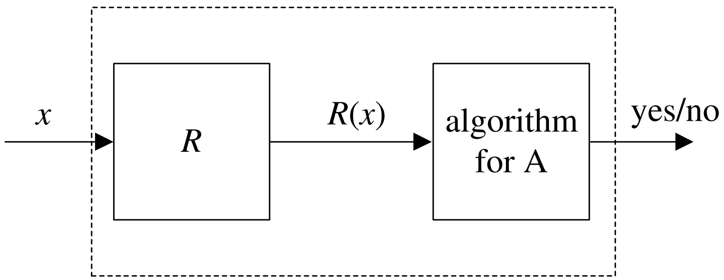
Degrees of Difficulty

- When is a problem more difficult than another?
- B **reduces to** A if there is a transformation R which for every input x of B yields an equivalent input $R(x)$ of A.
  - The answer to x for B is the same as the answer to $R(x)$ for A.
  - There must be restrictions on the complexity of computing R.
  - Otherwise, $R(x)$ might as well solve B.

Degrees of Difficulty (concluded)

- Problem A is at least as hard as problem B if B reduces to A.
- This makes intuitive sense: If A is able to solve your problem B, then A must be at least as powerful.

Reduction



Solving problem B by calling the algorithm for problem *once* and *without* further processing its answer.

Comments[a]

- Suppose B reduces to A via a transformation R.
- The input x is an instance of B.
- The output $R(x)$ is an instance of A.
- $R(x)$ may not span all possible instances of A.
- So some instances of A may never appear in the reduction.

[a]Contributed by Mr. Ming-Feng Tsai (D92922003) on October 29, 2003.

Reduction between Languages

- Language L_1 is **reducible to** L_2 if there is a function R computable by a deterministic TM in space $O(\log n)$.
- Furthermore, for all inputs x, $x \in L_1$ if and only if $R(x) \in L_2$.
- R is said to be a (**Karp**) **reduction** from L_1 to L_2.
- Note that by Theorem 21 (p. 180), R runs in polynomial time.
- If R is a reduction from L_1 to L_2, then $R(x) \in L_2$ is a legitimate algorithm for $x \in L_1$.

A Paradox?

- Degree of difficulty is not defined in terms of *absolute* complexity.
- So a language B $\in$ TIME(n^{99}) may be "easier" than a language A $\in$ TIME(n^3).
- This happens when B is reducible to A.
- But isn't this a contradiction when B $\notin$ TIME(n^k) for any $k < 99$?
- That is, how can a problem *requiring* n^{33} time be reducible to a problem solvable in n^3 time?

A Paradox? (concluded)

- The so-called contradiction is more apparent than real.
- When we solve the problem "$x \in$ B?" with "$R(x) \in$ A?", we must consider the time spent by $R(x)$ and its length $|\, R(x) \,|$.
- If $|\, R(x) \,| = \Omega(n^{33})$, then the time of answering "$R(x) \in$ A?" becomes $\Omega((n^{33})^3) = \Omega(n^{99})$.
- Suppose, on the other hand, that $|\, R(x) \,| = o(n^{33})$.
- Then $R(x)$ must run in time $\Omega(n^{99})$.
- In either case, there is no contradiction.

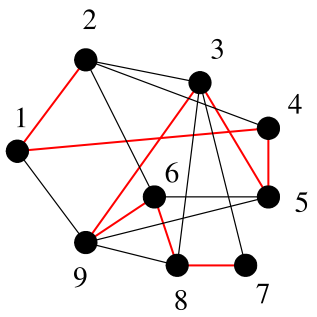
HAMILTONIAN PATH

- A **Hamiltonian path** of a graph is a path that visits every node of the graph exactly once.
- Suppose graph G has n nodes: $1, 2, \ldots, n$.
- A Hamiltonian path can be expressed as a permutation π of $\{\, 1, 2, \ldots, n \,\}$ such that
  - $\pi(i) = j$ means the ith position is occupied by node j.
  - $(\pi(i), \pi(i+1)) \in G$ for $i = 1, 2, \ldots, n-1$.
- HAMILTONIAN PATH asks if a graph has a Hamiltonian path.

Reduction of HAMILTONIAN PATH to SAT

- Given a graph G, we shall construct a CNF $R(G)$ such that $R(G)$ is satisfiable if and only if G has a Hamiltonian path.
- $R(G)$ has n^2 boolean variables x_{ij}, $1 \leq i, j \leq n$.
- x_{ij} means

  the ith position in the Hamiltonian path is occupied by node j.



$x_{12} = x_{21} = x_{34} = x_{45} = x_{53} = x_{69} = x_{76} = x_{88} = x_{97} = 1.$

The Clauses of $R(G)$ and Their Intended Meanings

1. Each node j must appear in the path.
   - $x_{1j} \vee x_{2j} \vee \cdots \vee x_{nj}$ for each j.
2. No node j appears twice in the path.
   - $\neg x_{ij} \vee \neg x_{kj}$ for all i, j, k with $i \neq k$.
3. Every position i on the path must be occupied.
   - $x_{i1} \vee x_{i2} \vee \cdots \vee x_{in}$ for each i.
4. No two nodes j and k occupy the same position in the path.
   - $\neg x_{ij} \vee \neg x_{ik}$ for all i, j, k with $j \neq k$.
5. Nonadjacent nodes i and j cannot be adjacent in the path.
   - $\neg x_{ki} \vee \neg x_{k+1,j}$ for all $(i, j) \notin G$ and $k = 1, 2, \ldots, n-1$.

The Proof

- $R(G)$ contains $O(n^3)$ clauses.
- $R(G)$ can be computed efficiently (simple exercise).
- Suppose $T \models R(G)$.
- From Clauses of 1 and 2, for each node j there is a unique position i such that $T \models x_{ij}$.
- From Clauses of 3 and 4, for each position i there is a unique node j such that $T \models x_{ij}$.
- So there is a permutation π of the nodes such that $\pi(i) = j$ if and only if $T \models x_{ij}$.

The Proof (concluded)

- Clauses of 5 furthermore guarantees that $(\pi(1), \pi(2), \ldots, \pi(n))$ is a Hamiltonian path.
- Conversely, suppose G has a Hamiltonian path

$$(\pi(1), \pi(2), \ldots, \pi(n)),$$

  where π is a permutation.
- Clearly, the truth assignment

$$T(x_{ij}) = \texttt{true} \text{ if and only if } \pi(i) = j$$

  satisfies all clauses of $R(G)$.