Computational Theory

Time Complexity

Curtis Larsen

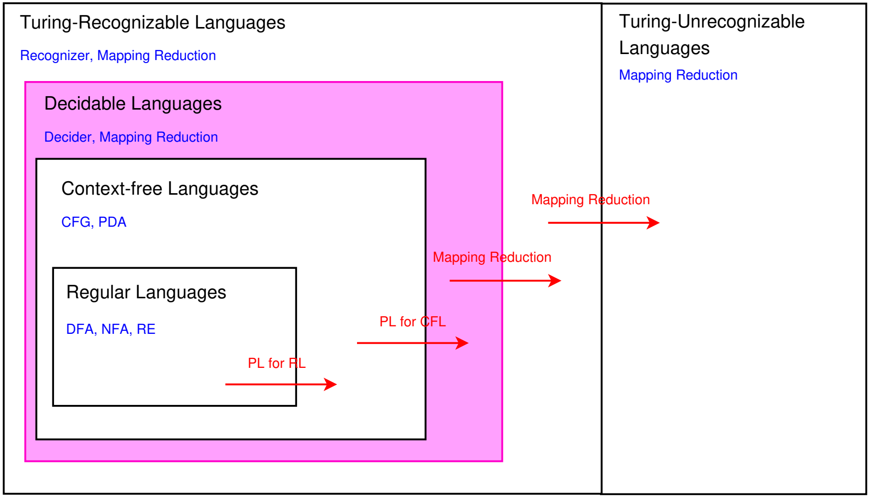
Utah Tech University—Computing

Fall 2024

Measuring Complexity

Reading: Sipser §7.1.

Language Landscape



We are only concerned with Decidable languages.

Definition 7.1

Let M be a deterministic Turing machine that halts on all inputs. The ***running time*** or ***time complexity*** of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on any input of length n. If $f(n)$ is the running time of M, we say that M runs in time $f(n)$ and that M is an $f(n)$ time Turing machine. Customarily we use n to represent the length of the input.

Definition 7.2

Big-O

Let f and g be functions $f, g : \mathbb{N} \to \mathbb{R}^+$. Say that $\boldsymbol{f(n) = O(g(n))}$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$,

$$f(n) \leq c \cdot g(n)$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an ***upper bound***. for $f(n)$, or more precisely, that $g(n)$. is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.

Definition 7.5

Small-o

Let f and g be functions $f, g : \mathbb{N} \to \mathbb{R}^+$. Say that $\boldsymbol{f(n) = o(g(n))}$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

In other words, $\boldsymbol{f(n) = o(g(n))}$ means that for any real number $c > 0$, a number n_0 exists, such that for every integer $n \geq n_0$.

$$f(n) < c \cdot g(n).$$

An Example Language

$$\mathsf{A} = \{0^k 1^k | k \geq 0\}$$

An Example Language

$$\mathsf{A} = \{0^k 1^k | k \geq 0\}$$

A decider for A.

Let $M_1 =$ “On input string w:

```
1. Scan across the tape, *reject* if a $0$ is found to the right of a $1$.
2. Repeat if both $0$s and $1$s are remain on the tape:
3.     Scan across the tape, crossing off a single $0$ and a single $1$.
4. If $0$s or $1s$ still remain on the tape, *reject*; Otherwise, *accept*.”
```

Analyzing Complexity

Let $M_1 =$ “On input string w:

1. Scan across the tape, *reject* if a 0 is found to the right of a 1.

Analyzing Complexity

Let $M_1 =$ "On input string w:

1. Scan across the tape, *reject* if a 0 is found to the right of a 1. $O(n)$
2. Repeat if both 0s and 1s are remain on the tape:

Analyzing Complexity

Let $M_1 =$ "On input string w:

1. Scan across the tape, *reject* if a 0 is found to the right of a 1. $O(n)$
2. Repeat if both 0s and 1s are remain on the tape: $O(n)$
3. Scan across the tape, crossing off a single 0 and a single 1.

Analyzing Complexity

Let $M_1 =$ "On input string w:

1. Scan across the tape, *reject* if a 0 is found to the right of a 1. $O(n)$
2. Repeat if both 0s and 1s are remain on the tape: $O(n)$
3. Scan across the tape, crossing off a single 0 and a single 1. $O(n)$
4. If 0s or $1s$ still remain on the tape, *reject*; Otherwise, *accept*."

Analyzing Complexity

Let $M_1 =$ "On input string w:

1. Scan across the tape, *reject* if a 0 is found to the right of a 1. $O(n)$
2. Repeat if both 0s and 1s are remain on the tape: $O(n)$
3.   Scan across the tape, crossing off a single 0 and a single 1. $O(n)$
4. If 0s or $1s$ still remain on the tape, *reject*; Otherwise, *accept*."$O(n)$

Analyzing Complexity

Let $M_1 =$ “On input string w:

1. Scan across the tape, *reject* if a 0 is found to the right of a 1. $O(n)$
2. Repeat if both 0s and 1s are remain on the tape: $O(n)$
3. Scan across the tape, crossing off a single 0 and a single 1. $O(n)$
4. If 0s or $1s$ still remain on the tape, *reject*; Otherwise, *accept*.”$O(n)$

Total running time:

Analyzing Complexity

Let $M_1 =$ "On input string w:

1. Scan across the tape, *reject* if a 0 is found to the right of a 1. $O(n)$
2. Repeat if both 0s and 1s are remain on the tape: $O(n)$
3. Scan across the tape, crossing off a single 0 and a single 1. $O(n)$
4. If 0s or $1s$ still remain on the tape, *reject*; Otherwise, *accept*."$O(n)$

Total running time: $O(n) + O(n)O(n) + O(n) = O(n^2)$.

Definition 7.7

Let $t : \mathbb{N} \to \mathbb{R}^+$. Define the ***time complexity class***, **TIME**$(t(n))$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

Example Language

Because the complexity of M_1 is $O(n^2)$, we can say that A $\in$ TIME(n^2).

An Example Language

$$\mathsf{A} = \{0^k 1^k | k \geq 0\}$$

An Example Language

$$\mathsf{A} = \{0^k 1^k | k \geq 0\}$$

Another decider for A. This decider is a two-tape deterministic Turing machine.

An Example Language

$$\text{A} = \{0^k1^k | k \geq 0\}$$

Another decider for A. This decider is a two-tape deterministic Turing machine.

Let $M_3 =$ “On input string w:

1. Scan across tape 1, and *reject* if a 0 is found to the right of a 1.
2. Scan across the 0s on tape 1, until the first 1, copying the 0s onto tape 2.
3. Scan across the 1s on tape 1, until the end of input. For each 1 read on tape 1, cross of a 0 on tape 2. If all 0s are crossed off before all the 1s are read, *reject*.
4. If all the 0s have been crossed off, *accept*; otherwise, *reject*.”

Analyzing Complexity

Another decider for A. This decider is a two-tape deterministic Turing machine.

Let $M_3 =$ “On input string w:

1. Scan across tape 1, and *reject* if a 0 is found to the right of a 1.

Analyzing Complexity

Another decider for A. This decider is a two-tape deterministic Turing machine.

Let $M_3 =$ “On input string w:

1. Scan across tape 1, and *reject* if a 0 is found to the right of a 1. $O(n)$
2. Scan across the 0s on tape 1, until the first 1, copying the 0s onto tape 2.

Analyzing Complexity

Another decider for A. This decider is a two-tape deterministic Turing machine.

Let $M_3 =$ “On input string w:

1. Scan across tape 1, and *reject* if a 0 is found to the right of a 1. $O(n)$
2. Scan across the 0s on tape 1, until the first 1, copying the 0s onto tape 2. $O(n)$
3. Scan across the 1s on tape 1, until the end of input. For each 1 read on tape 1, cross of a 0 on tape 2. If all 0s are crossed off before all the 1s are read, *reject*.

Analyzing Complexity

Another decider for A. This decider is a two-tape deterministic Turing machine.

Let $M_3 =$ “On input string w:

1. Scan across tape 1, and *reject* if a 0 is found to the right of a 1. $O(n)$
2. Scan across the 0s on tape 1, until the first 1, copying the 0s onto tape 2. $O(n)$
3. Scan across the 1s on tape 1, until the end of input. For each 1 read on tape 1, cross of a 0 on tape 2. If all 0s are crossed off before all the 1s are read, *reject*. $O(n)$
4. If all the 0s have been crossed off, *accept*; otherwise, *reject*.”

Analyzing Complexity

Another decider for A. This decider is a two-tape deterministic Turing machine.

Let $M_3 =$ "On input string w:

1. Scan across tape 1, and *reject* if a 0 is found to the right of a 1. $O(n)$
2. Scan across the 0s on tape 1, until the first 1, copying the 0s onto tape 2. $O(n)$
3. Scan across the 1s on tape 1, until the end of input. For each 1 read on tape 1, cross of a 0 on tape 2. If all 0s are crossed off before all the 1s are read, *reject*. $O(n)$
4. If all the 0s have been crossed off, *accept*; otherwise, *reject*." $O(n)$

Analyzing Complexity

Another decider for A. This decider is a two-tape deterministic Turing machine.

Let $M_3 =$ “On input string w:

1. Scan across tape 1, and *reject* if a 0 is found to the right of a 1. $O(n)$
2. Scan across the 0s on tape 1, until the first 1, copying the 0s onto tape 2. $O(n)$
3. Scan across the 1s on tape 1, until the end of input. For each 1 read on tape 1, cross of a 0 on tape 2. If all 0s are crossed off before all the 1s are read, *reject*. $O(n)$
4. If all the 0s have been crossed off, *accept*; otherwise, *reject*.” $O(n)$

Total running time:

Analyzing Complexity

Another decider for A. This decider is a two-tape deterministic Turing machine.

Let $M_3 =$ "On input string w:

1. Scan across tape 1, and *reject* if a 0 is found to the right of a 1. $O(n)$
2. Scan across the 0s on tape 1, until the first 1, copying the 0s onto tape 2. $O(n)$
3. Scan across the 1s on tape 1, until the end of input. For each 1 read on tape 1, cross of a 0 on tape 2. If all 0s are crossed off before all the 1s are read, *reject*. $O(n)$
4. If all the 0s have been crossed off, *accept*; otherwise, *reject*." $O(n)$

Total running time: $O(n) + O(n) + O(n) + O(n) = O(n)$.

Complexity Class

Because the complexity of M_3 is $O(n)$, we can now say that A $\in$ TIME(n).

Can we still say A $\in$ TIME(n^2)?

Complexity Class

Because the complexity of M_3 is $O(n)$, we can now say that A $\in$ TIME(n).

Can we still say A $\in$ TIME(n^2)? Yes

Theorem 7.8

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multi-tape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

Theorem 7.8

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multi-tape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

Proof Idea: Recall the single-tape Turing machine that simulates a multi-tape Turing machine from Theorem 3.13. Each step on the multi-tape machine can take at most $O(t(n))$ steps in the single-tape simulator, producing the total run time of $O(t^2(n))$.

Definition 7.9

Let N be a nondeterministic Turing machine that is a decider. The ***running time*** of N is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on any input of length n.

Definition 7.9

Let N be a nondeterministic Turing machine that is a decider. The ***running time*** of N is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on any input of length n.

This deserves a good diagram.

Theorem 7.11

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

Theorem 7.11

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

Proof Idea: Recall the deterministic Turing machine that simulates a nondeterministic Turing machine from Theorem 3.16. The nondeterministic computation tree has height of $t(n)$ and at most $b^{O(t(n))}$ leaves, where b is the branching factor. The simulation does breadth first search of this tree. This has run time of $O(b^{O(t(n))})$, which is $2^{O(t(n))}$.

Computational Models

In computability theory: the computational model does not matter.

Computational Models

In computability theory: the computational model does not matter.

In complexity theory: the computational model does matter.

Important Results

- Big-O
- Time Complexity Classes: TIME$(t(n))$
- Theorem 7.8 $t(n)$ multi-tape Turing machines have equivalent $O(t^2(n))$ single-tape Turing machines.
- Theorem 7.11 $t(n)$ non-deterministic Turing machines have equivalent $2^{O(t(n))}$ deterministic single-tape Turing machines.
- Polynomial time, $t(n) = O(n^k)$, is “easy”.
- Exponential time, $t(n) = O(k^n)$, is “hard”.

The Class P

Reading: Sipser §7.2.

Definition 7.12

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

$$\mathsf{P} = \bigcup_k \mathsf{TIME}(n^k).$$

Graphs

A graph is a collection of vertices (or nodes) and edges. With each edge connecting a pair of nodes.

In directed graphs, edges are directional, and represented as an ordered pair (a, b). In undirected graphs, edges are bi-directional, and represented as an unordered pair $\{a, b\}$.

When analyzing the time complexity of graph algorithms, we often use the number of vertices as the size of the graph. If we want to be more detailed, we account for the number of edges as well.

$G = (V, E)$, where V is the set of vertices and E is the set of edges. In a fully connected graph, $|E| = |V|^2$. In general, $|E| \leq |V|^2$. $|G| = O(|V| + |E|) = O(|V|^2)$.

PATH

A graph problem:

$$\text{PATH} = \{\langle G, s, t\rangle | G \text{ is a directed graph that has a directed path from node } s \text{ to node } t\}$$

Theorem 7.14

PATH $\in$ P

How could we prove this?

Theorem 7.14

PATH $\in$ P

How could we prove this?

1. Provide a decider for PATH, M_{PATH}.
2. Analyze the time complexity of M_{PATH}.
3. If M_{PATH}'s time complexity is $O(n^k)$ and $k \in \mathbb{N}$, then PATH $\in$ TIME$(n^k) \in$ P.

PATH

Let $M_{\text{PATH}} =$ "On input $\langle G, s, t \rangle$:

1. Place a mark on node s.

PATH

Let $M_{\text{PATH}} =$ "On input $\langle G, s, t \rangle$:

1. Place a mark on node s.
2. Repeat the following until no additional nodes are marked:

PATH

Let M_{PATH} = "On input $\langle G, s, t \rangle$:

```
1. Place a mark on node $s$.
2. Repeat the following until no additional nodes are marked:
3.     Scan all edges of $G$. If an edge $(a, b)$ is found going from a
       marked node $a$ to an unmarked node $b$, mark node $b$.
```

PATH

Let $M_{\text{PATH}} =$ "On input $\langle G, s, t \rangle$:

```
1. Place a mark on node $s$.
2. Repeat the following until no additional nodes are marked:
3.     Scan all edges of $G$. If an edge $(a, b)$ is found going from a
       marked node $a$ to an unmarked node $b$, mark node $b$.
4. If $t$ is marked, *accept*. Otherwise, *reject*."
```

PATH

Let $M_{\text{PATH}} =$ "On input $\langle G, s, t\rangle$:

```
1. Place a mark on node $s$.
2. Repeat the following until no additional nodes are marked:
3.     Scan all edges of $G$. If an edge $(a, b)$ is found going from a
       marked node $a$ to an unmarked node $b$, mark node $b$.
4. If $t$ is marked, accept. Otherwise, reject."
```

The loop will terminate after at most $|V||E| \leq |V|^3$ edge checks. M_{PATH} halts. If there is a path from s to t in G, M_{PATH} will accept. Otherwise, M_{PATH} will reject. M_{PATH} is a decider for PATH.

What is the time complexity class of PATH?

PATH

Let $M_{\text{PATH}} =$ "On input $\langle G, s, t \rangle$:

```
1. Place a mark on node $s$.
2. Repeat the following until no additional nodes are marked:
3.     Scan all edges of $G$. If an edge $(a, b)$ is found going from a
       marked node $a$ to an unmarked node $b$, mark node $b$.
4. If $t$ is marked, *accept*. Otherwise, *reject*."
```

The loop will terminate after at most $|V||E| \leq |V|^3$ edge checks. M_{PATH} halts. If there is a path from s to t in G, M_{PATH} will accept. Otherwise, M_{PATH} will reject. M_{PATH} is a decider for PATH.

What is the time complexity class of PATH? TIME$(|V||E|) \in$ P

Encoding of Numbers

When a number is an input, what is the size of the input?

Encoding of Numbers

When a number is an input, what is the size of the input?

It depends on the representation used: $\langle n \rangle$.

Encoding of Numbers

When a number is an input, what is the size of the input?

It depends on the representation used: $\langle n \rangle$.

Unary notation for encoding uses n 1's to represent n. (e.g. $\langle 5 \rangle = 11111$). $|n| = n$.

Encoding of Numbers

When a number is an input, what is the size of the input?

It depends on the representation used: $\langle n \rangle$.

Unary notation for encoding uses n 1's to represent n. (e.g. $\langle 5 \rangle = 11111$). $|n| = n$.

Base k notation, with $k \geq 2$, is much more compact. (e.g. base 2, $\langle 5 \rangle = 101$). $|n| = \lceil log_k(n) \rceil$. This is much better.

RELPRIME

Definition: Two numbers are *relatively prime* if 1 is the largest integer that evenly divides them both. (e.g. 10 and 21 are not prime, but are relatively prime.)

Another language definition:

$$\text{RELPRIME} = \{\langle x, y\rangle | x \text{ and } y \text{ are relatively prime }\}.$$

Theorem 7.15

RELPRIME $\in$ P

How could we prove this?

Theorem 7.15

RELPRIME $\in$ P

How could we prove this?

1. Provide a decider for RELPRIME, M_{RELPRIME}.
2. Analyze the time complexity of M_{RELPRIME}.
3. If M_{RELPRIME}'s time complexity is $O(n^k)$ and $k \in \mathbb{N}$, then RELPRIME $\in$ TIME$(n^k) \in$ P.

RELPRIME

The Euclidean algorithm computes the greatest common divisor of two natural numbers:

Let $E =$ "On input $\langle x, y \rangle$:

```
1. Repeat until $y = 0$:
2.     Assign $x \leftarrow x \mod y$.
3.     Exchange $x$ and $y$.
4. Output $x$."
```

RELPRIME

The Euclidean algorithm computes the greatest common divisor of two natural numbers:

Let $E =$ “On input $\langle x, y \rangle$:

```
1. Repeat until $y = 0$:
2.     Assign $x \leftarrow x \mod y$.
3.     Exchange $x$ and $y$.
4. Output $x$.”
```

We accept its correctness from numerous textbooks. The loop in E causes x to lose at least 1 bit, before swapping it with y. Every two iterations the loop will cause both of the numbers to lose at least one bit. The loop in E will terminate after at most 2 min($log_2(x)$, $log_2(y)$) repetitions.

RELPRIME

Let $M_{\text{RELPRIME}} =$ "On input $\langle x, y \rangle$:

1. Run E on x and y.
2. If the result is 1, *accept*. Otherwise, *reject*."

RELPRIME

Let M_{RELPRIME} = "On input $\langle x, y \rangle$:

1. Run E on x and y.
2. If the result is 1, *accept*. Otherwise, *reject*."

M_{RELPRIME} halts, because E halts. M_{RELPRIME} is a decider for RELPRIME.

RELPRIME

Let M_{RELPRIME} = "On input $\langle x, y \rangle$:

1. Run E on x and y.
2. If the result is 1, *accept*. Otherwise, *reject*."

M_{RELPRIME} halts, because E halts. M_{RELPRIME} is a decider for RELPRIME.

What is the size of the input?

RELPRIME

Let M_{RELPRIME} = "On input $\langle x, y \rangle$:

1. Run E on x and y.
2. If the result is 1, *accept*. Otherwise, *reject*."

M_{RELPRIME} halts, because E halts. M_{RELPRIME} is a decider for RELPRIME.

What is the size of the input? $n = \lceil log_2(x) \rceil + \lceil log_2(y) \rceil$

What is the time complexity class of RELPRIME?

RELPRIME

Let $M_{\text{RELPRIME}} =$ "On input $\langle x, y \rangle$:

1. Run E on x and y.
2. If the result is 1, *accept*. Otherwise, *reject*."

M_{RELPRIME} halts, because E halts. M_{RELPRIME} is a decider for RELPRIME.

What is the size of the input? $n = \lceil log_2(x) \rceil + \lceil log_2(y) \rceil$

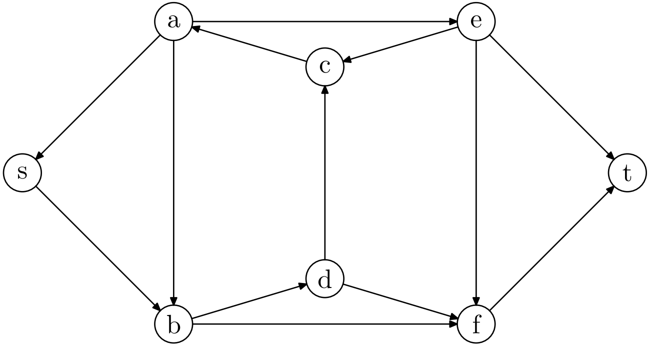
What is the time complexity class of RELPRIME? TIME$(n) \in$ P

The Class NP

Reading: Sipser §7.3.

Hamiltonian Paths

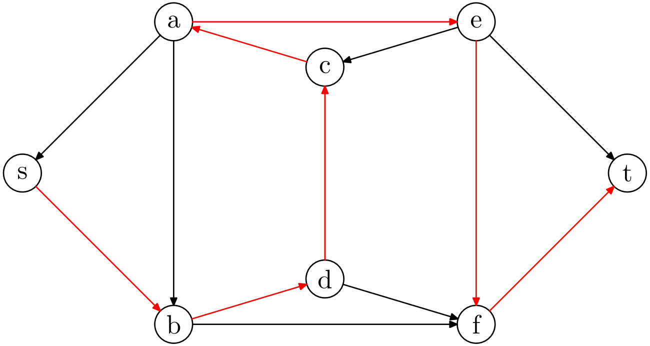
A Hamiltonian Path is a directed path in a graph that visits each node exactly once.



What is a Hamiltonian Path in this graph, starting at s and ending at t?

Hamiltonian Paths

A Hamiltonian Path is a directed path in a graph that visits each node exactly once.


$$\begin{tikzcd}
 & a \arrow[ld] \arrow[dd] \arrow[rr, red] & & e \arrow[ld] \arrow[dd, red] \arrow[rd] & \\
s \arrow[rd, red] & & c \arrow[lu, red] & & t \\
 & b \arrow[r, red] \arrow[rr] & d \arrow[u, red] \arrow[r] & f \arrow[ru, red] &
\end{tikzcd}$$

HAMPATH

Another graph problem:

$$\text{HAMPATH} = \{\langle G, s, t\rangle | G \text{ is a directed graph that has a Hamiltonian path from node } s \text{ to node } t\}$$

HAMPATH

Another graph problem:

$$\text{HAMPATH} = \{\langle G, s, t\rangle | G \text{ is a directed graph that has a Hamiltonian path from node } s \text{ to node } t\}$$

Is HAMPATH in P?

HAMPATH

We don’t know if HAMPATH is in P. We can prove a weaker property of HAMPATH:

HAMPATH

We don't know if HAMPATH is in P. We can prove a weaker property of HAMPATH:

Given a path c, described as an ordered list of nodes, in G, we can verify if it is a Hamiltonian path in polynomial time. We provide a *verifier* that completes in polynomial time.

HAMPATH

Let V_{HAMPATH} = "On input $\langle\langle G, s, t\rangle, c\rangle$:

```
1.  Mark all nodes $v$ in $G$ as unvisited.
2.  Let $u$ be the first node in $c$.
3.  If $u \neq s$, then *reject*.
4.  Mark $u$ as visited.
5.  For each $v$ in $c$, starting with the second node:
6.      If $(u, v) \notin E$, then *reject*.
7.      If $v$ is visited, then *reject*.
8.      Mark $v$ as visited.
9.      $u = v$.
10. If $u \neq t$, then *reject*.
11. If any node is not visited, then *reject*.
12. *accept*.
```

COMPOSITES

Another number problem:

$$\text{COMPOSITES} = \{\langle x\rangle | x = pq, \text{ for integers } p, q > 1\}$$

COMPOSITES

Another number problem:

$$\text{COMPOSITES} = \{\langle x\rangle | x = pq,\ \text{for integers } p, q > 1\}$$

Is COMPOSITES in P?

COMPOSITES

In recent years, a proof that COMPOSITES is in P has been given. We will not prove that here. Instead, we will prove the weaker property that COMPOSITES is *polynomially verifiable*.

COMPOSITES

In recent years, a proof that COMPOSITES is in P has been given. We will not prove that here. Instead, we will prove the weaker property that COMPOSITES is *polynomially verifiable*.

Given c, in the form of two numbers p, q, we can verify if their product is x, in polynomial time. We provide a *verifier* that completes in polynomial time.

COMPOSITES

In recent years, a proof that COMPOSITES is in P has been given. We will not prove that here. Instead, we will prove the weaker property that COMPOSITES is *polynomially verifiable*.

Given c, in the form of two numbers p, q, we can verify if their product is x, in polynomial time. We provide a *verifier* that completes in polynomial time.

Let $V_{\text{COMPOSITES}} =$ "On input $\langle\langle x\rangle, c\rangle$:

1. Let $p, q = c$.
2. Multiply p and q.
3. If the product is not x, then *reject*.
4. *accept*.

Definition 7.18

A ***verifier*** for a language A is an algorithm V, where

$$\text{A} = \{w | \text{V accepts } \langle w, c\rangle \text{ for some string } c\}.$$

We measure the time of a verifier only in terms of the length of w, so a ***polynomial time verifier*** runs in polynomial time in the length of w. Language A is ***polynomially verifiable*** if it has a polynomial time verifier.

c is called the ***certificate***, or ***proof***, of membership.

Definition 7.19

NP is the class of languages that have polynomial time verifiers.

Definition 7.19

NP is the class of languages that have polynomial time verifiers.

What does NP stand for?

Definition 7.19

NP is the class of languages that have polynomial time verifiers.

What does NP stand for? ***nondeterministic polynomial time***.

Definition 7.19

NP is the class of languages that have polynomial time verifiers.

What does NP stand for? ***nondeterministic polynomial time***.

Why is P $\subseteq$ NP?

Definition 7.19

NP is the class of languages that have polynomial time verifiers.

What does NP stand for? ***nondeterministic polynomial time***.

Why is P $\subseteq$ NP?

If a problem is solvable in polynomial time, then the verifier could solve the problem, and check if the certificate matches the solution, all in polynomial time.

Theorem 7.20

A language is in NP if and only if it is decided by some nondeterministic polynomial time Turing machine.

Theorem 7.20

A language is in NP if and only if it is decided by some nondeterministic polynomial time Turing machine.

Proof Idea: Convert a polynomial time verifier to an equivalent polynomial time Nondeterministic Turing Machine, and vice versa.

Theorem 7.20

Proof (part1): Let $A \in$ NP, with V a polynomial time verifier for A, which exists because A is in NP. V runs in time $O(n^k)$.

Let $N =$ “On input w of length n:

1. Non-deterministically select string c of length at most n^k.
2. Run V on input $\langle w, c\rangle$.
3. If V accepts, *accept*; otherwise, *reject*.”

If A is in NP, then N is a polynomial time Nondeterministic decider.

Theorem 7.20

Proof (part2): Assume that A is decided by a polynomial time non-deterministic Turing machine, N. Construct the following verifier, V.

Let $V =$ "On input $\langle w, c \rangle$ where w and c are strings:

1. Simulate N on input w, treating each symbol of c as a description of the non-deterministic choice to make at each step.
2. If this branch of N's computation accepts, *accept*; otherwise, *reject*."

If A is in decided by a polynomial time Nondeterministic decider then V is a verifier for A.

Definition 7.21

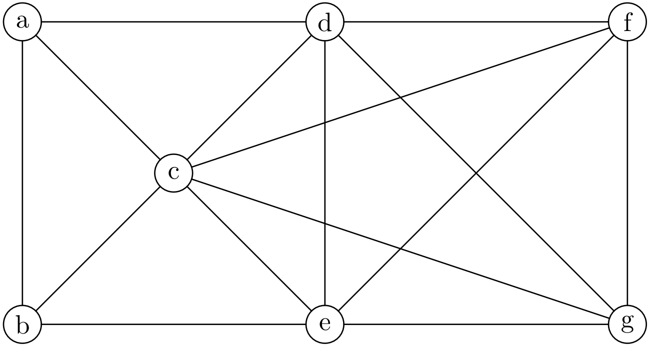
NTIME$(t(n)) = \{L | L$ is a language decided by a $O(t(n))$ time nondeterministic Turing machine $\}$.

Corollary 7.22

$$\mathsf{NP} = \bigcup_k \mathsf{NTIME}(n^k).$$

Cliques in Graphs

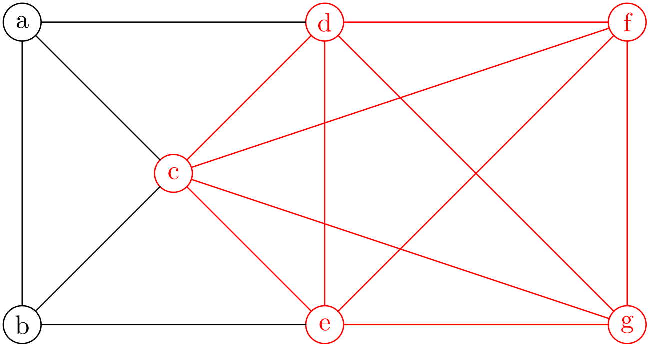
A clique in an undirected graph is a subgraph, where all pairs of graphs are connected by an edge.



What nodes are in the 5-clique?

Cliques in Graphs

A clique in an undirected graph is a subgraph, where all pairs of graphs are connected by an edge.



What nodes are in the 5-clique? $\{c, d, e, f, g\}$

CLIQUE

Another graph problem:

$$\text{CLIQUE} = \{\langle G, k\rangle | G \text{ is an undirected graph that has a } k\text{-clique}\}$$

CLIQUE

Another graph problem:

$$\text{CLIQUE} = \{\langle G, k\rangle | G \text{ is an undirected graph that has a } k\text{-clique}\}$$

Is CLIQUE in NP?

Theorem 7.24

CLIQUE $\in$ NP

Theorem 7.24

CLIQUE $\in$ NP

Proof Idea: A certificate for CLIQUE is a list of nodes in a clique.

Theorem 7.24

CLIQUE $\in$ NP

Proof Idea: A certificate for CLIQUE is a list of nodes in a clique.

Proof:

Let $V =$ "On input $\langle\langle G, k\rangle, c\rangle$:

1. Test whether $|c| = k$.
2. Test whether all nodes in c are in G.
3. Test whether all pairs of nodes in c have connecting edges in G.
4. If all tests pass, *accept*; otherwise *reject*."

V runs in time polynomial in the size of G, and decides if c is a k-clique of G.

SUBSET-SUM

Consider this numeric problem:

Given: A set S of k numbers $x_1, ..., x_k$ and a number t.

Find: A subset $y_1, ..., y_m$ of S such that $\sum_i y_i = t$.

SUBSET-SUM

Consider this numeric problem:

Given: A set S of k numbers $x_1, ..., x_k$ and a number t.

Find: A subset $y_1, ..., y_m$ of S such that $\sum_i y_i = t$.

As a language:

$$\text{SUBSET-SUM} = \{\langle S, t\rangle | S = \{x_1, ..., x_k\} \text{ and for some } \{y_1, ..., y_m\} \subseteq \{x_1, ..., x_k\} \text{ we have } \sum_i y_i = t\}$$

SUBSET-SUM

Consider this numeric problem:

Given: A set S of k numbers $x_1, ..., x_k$ and a number t.

Find: A subset $y_1, ..., y_m$ of S such that $\sum_i y_i = t$.

As a language:

$$\text{SUBSET-SUM} = \{\langle S, t\rangle | S = \{x_1, ..., x_k\} \text{ and for some } \{y_1, ..., y_m\} \subseteq \{x_1, ..., x_k\} \text{ we have } \sum_i y_i = t\}$$

Is SUBSET-SUM in NP?

Theorem 7.25

SUBSET-SUM $\in$ NP

Theorem 7.25

SUBSET-SUM $\in$ NP

Proof Idea: A certificate for SUBSET-SUM is a set of numbers.

Theorem 7.25

SUBSET-SUM $\in$ NP

Proof Idea: A certificate for SUBSET-SUM is a set of numbers.

Proof:

Let $V =$ "On input $\langle\langle S, t\rangle, c\rangle$:

1. Test whether the numbers in c add up to t.
2. Test whether all numbers in c are in S.
3. If all tests pass, *accept*; otherwise *reject*."

V runs in time polynomial in the size of S, and decides if c is a subset of S that sums to t.

coNP

Are these languages in NP?

- $\overline{\text{HAMPATH}}$
- $\overline{\text{CLIQUE}}$
- $\overline{\text{SUBSET-SUM}}$

coNP

Are these languages in NP?

- $\overline{\text{HAMPATH}}$
- $\overline{\text{CLIQUE}}$
- $\overline{\text{SUBSET-SUM}}$

It's not obvious. Verifying something is not present appears to be more difficult than verifying that it is present.

coNP

Are these languages in NP?

- $\overline{\text{HAMPATH}}$
- $\overline{\text{CLIQUE}}$
- $\overline{\text{SUBSET-SUM}}$

It's not obvious. Verifying something is not present appears to be more difficult than verifying that it is present.

Let coNP be the complexity class of languages that are complements of languages in NP.

P vs NP

P is the class of languages for which membership can be *decided* quickly.

NP is the class of languages for which membership can be *verified* quickly.

P vs NP

P is the class of languages for which membership can be *decided* quickly.

NP is the class of languages for which membership can be *verified* quickly.

The big question: P $=$ NP or P $\subset$ NP?

P vs NP

P is the class of languages for which membership can be *decided* quickly.

NP is the class of languages for which membership can be *verified* quickly.

The big question: $\mathrm{P} = \mathrm{NP}$ or $\mathrm{P} \subset \mathrm{NP}$?

The answer is not known.

P vs NP

P is the class of languages for which membership can be *decided* quickly.

NP is the class of languages for which membership can be *verified* quickly.

The big question: P $=$ NP or P $\subset$ NP?

The answer is not known.

We can prove:

$$\text{NP} \subseteq \text{EXPTIME} = \bigcup_k \text{TIME}(2^{n^k}).$$

NP-Completeness

Reading: Sipser §7.4.

Definition 7.28

A function $f : \Sigma^* \to \Sigma^*$ is a ***polynomial time computable function*** if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w.

Definition 7.29

Language A is ***polynomial time mapping reducible***, or ***polynomial time reducible***, to language B, written A $\leq_P$ B, if a polynomial time computable function $f : \Sigma^* \rightarrow \Sigma^*$ exists, where for every w,

$$w \in \text{A} \Leftrightarrow f(w) \in \text{B}.$$

The function f is called the ***polynomial time reduction*** of A to B.

Theorem 7.31

If A $\leq_P$ B and B $\in$ P, then A $\in$ P.

Theorem 7.31

If A $\leq_P$ B and B $\in$ P, then A $\in$ P.

Proof: Let M be the polynomial time decider for B and f be the polynomial time reduction from A to B.

Theorem 7.31

If A $\leq_P$ B and B $\in$ P, then A $\in$ P.

Proof: Let M be the polynomial time decider for B and f be the polynomial time reduction from A to B.

Let $N =$ "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

Theorem 7.31

If A $\leq_P$ B and B $\in$ P, then A $\in$ P.

Proof: Let M be the polynomial time decider for B and f be the polynomial time reduction from A to B.

Let $N =$ “On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs.”

Is $w \in$ A $\Leftrightarrow f(w) \in$ B?

Theorem 7.31

If A $\leq_P$ B and B $\in$ P, then A $\in$ P.

Proof: Let M be the polynomial time decider for B and f be the polynomial time reduction from A to B.

Let $N =$ “On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs.”

Is $w \in$ A $\Leftrightarrow f(w) \in$ B?

Is N polynomial time?

Reduction proof outline

A $\leq_P$ B

- Describe a generic instance of A.
- Describe a generic instance of B.
- Provide the polynomial time reduction function.
- Prove the function is polynomial time computable.
- Prove that *any* instance of A can be reduced to *some* instance of B.
- Prove that *any* non-instance of A can be reduced to *some* non-instance of B. **Alternatively:** Prove that *any* reachable instance of B can be only be reduced from *some* instance of A.
- Conclude that the conditions of polynomial time mapping reduction have been met.

3SAT

Given: Logic variables $x_1, x_2, ..., x_n$, disjunctive clauses $c_1, c_2, ..., c_m$ with 3 literals per clause, and ϕ the conjunction of all clauses. This is a "3 CNF-formula".

Find: Whether there is an assignment of truth values for all variables that satisfies ϕ.

Sample: $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$
Satisfying assignment: $x_1 = 0, x_2 = 1$.

3SAT

A satisfiability problem:

$$3\text{SAT} = \{\langle\phi\rangle | \phi \text{ is a satisfiable 3 CNF-formula } \}$$

Theorem 7.32

3SAT is polynomial time reducibile to CLIQUE.

3SAT

3SAT $= \{\langle\phi\rangle | \phi$ is a satsifiable 3 cnf-formula$\}$.

A 3 cnf-formula is a conjunctive normal form formula with 3 literals per clause.

CLIQUE

CLIQUE $= \{\langle G, k\rangle | G$ is an undirected graph with a k-clique$\}$.

A ***clique*** in an undirected graph is a subgraph, wherein every two nodes are connected by an edge. A ***k-clique*** is a clique that contains k nodes.

Example Polynomial Time Reduction

Reduction from 3SAT to CLIQUE.

Definition 7.34

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

Definition 7.xx

A language B is ***NP-hard*** if all problems in NP are polynomial time reducible to it, even though it may not be in NP itself.

Definition 7.yy (Alternate Form of NP-Completeness)

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. B is NP-HARD.

Theorem 7.35

If B is NP-complete and B $\in$ P, then P $=$ NP.

Proof:

Theorem 7.35

If B is NP-complete and B $\in$ P, then P $=$ NP.

Proof:

By definition 7.34, any problem in NP is polynomial time reducible to B. For any A $\in$ NP, let R_{AB} be the reduction from A to B. Let M_{B} be the polynomial time decider for B, guaranteed to exist by definition 7.12.

Theorem 7.35

If B is NP-complete and B $\in$ P, then P $=$ NP.

Proof:

By definition 7.34, any problem in NP is polynomial time reducible to B. For any A $\in$ NP, let R_{AB} be the reduction from A to B. Let M_{B} be the polynomial time decider for B, guaranteed to exist by definition 7.12.

Let $M_{\mathsf{A}} =$ "On input w_{A}, a possible member of A:

1. Run R_{AB} on w_{A} to compute w_{B}.
2. Run M_{B} on w_{B}. If M_{B} accepts, *accept*; otherwise *reject*."

Theorem 7.35

If B is NP-complete and B $\in$ P, then P $=$ NP.

Proof:

By definition 7.34, any problem in NP is polynomial time reducible to B. For any A $\in$ NP, let R_{AB} be the reduction from A to B. Let M_{B} be the polynomial time decider for B, guaranteed to exist by definition 7.12.

Let M_{A} $=$ "On input w_{A}, a possible member of A:

1. Run R_{AB} on w_{A} to compute w_{B}.
2. Run M_{B} on w_{B}. If M_{B} accepts, *accept*; otherwise *reject*."

Both steps are polynomial time. This machine can be used to solve any problem in NP in polynomial time. If B exists with the properties above, then P $=$ NP.

Theorem 7.36

If B is NP-complete and B $\leq_P$ C for C in NP, then C is NP-complete.

Proof idea:

Theorem 7.36

If B is NP-complete and B $\leq_P$ C for C in NP, then C is NP-complete.

Proof idea:

C is in NP, so we only need to prove it is also NP-HARD. By definition 7.34, all of NP polynomial time reduces to B. By the conditions above, B $\leq_P$ C. By serial application, we can polynomial time reduce any member of NP to C, making it NP-HARD.

NP-completeness Proof Process

To prove language B is NP-complete.

1. Prove B $\in$ NP.
   - Describe a certificate for B.
   - Provide a polynomial time verifier for B. Must include arguments for correctness and time complexity of verifier.
2. Prove B is NP-HARD.
   - Select a known NP-COMPLETE language, C.
   - Provide a polynomial time reduction from C to B.
     - Instance descriptions for both C and B.
     - Reduction process $f(w)$.
   - Prove reduction from C to B is polynomial.
   - Prove $w \in$ C $\Rightarrow f(w) \in$ B.
   - Prove $w \notin$ C $\Rightarrow f(w) \notin$ B ***or*** $f(w) \in$ B $\Rightarrow w \in$ C.
   - Conclude B is NP-HARD.
3. Conclude B is NP-COMPLETE.

Theorem 7.37

SAT is NP-complete.

Theorem 7.37

SAT is NP-complete.

Proof to come later.

DOMINATING-SET

DOMINATING-SET $= \{\langle G, k\rangle | G$ is an undirected graph that has a k-node dominating set $\}$.

A ***dominating set*** is a subset of nodes where every other node of G is adjacent to at least one of those nodes.

DOMINATING-SET

DOMINATING-SET $= \{\langle G, k\rangle | G$ is an undirected graph that has a k-node dominating set $\}$.

A ***dominating set*** is a subset of nodes where every other node of G is adjacent to at least one of those nodes.

What does a certificate for DOMINATING-SET look like?

NP-Complete Problems

Reading: Sipser §7.5.

VERTEX-COVER

VERTEX-COVER $= \{\langle G, k\rangle | G$ is an undirected graph that has a k-node vertex cover $\}$.

A ***vertex cover*** is a subset of nodes where every edge of G touches one of those nodes.

VERTEX-COVER

VERTEX-COVER $= \{\langle G, k\rangle | G$ is an undirected graph that has a k-node vertex cover $\}$.

A ***vertex cover*** is a subset of nodes where every edge of G touches one of those nodes.

What does a certificate for VERTEX-COVER look like?

Theorem 7.44

VERTEX-COVER is NP-complete.

Theorem 7.44

VERTEX-COVER is NP-complete.

Proof:

Theorem 7.44

VERTEX-COVER is NP-complete.

Proof:

1. VERTEX-COVER is in NP.
   How to prove?

Theorem 7.44

VERTEX-COVER is NP-complete.

Proof:

1. VERTEX-COVER is in NP.
   How to prove? Provide a verifier.

Theorem 7.44

VERTEX-COVER is NP-complete.

Proof:

1. VERTEX-COVER is in NP.
   How to prove? Provide a verifier.
2. VERTEX-COVER is NP-HARD.
   How to prove?

Theorem 7.44

VERTEX-COVER is NP-complete.

Proof:

1. VERTEX-COVER is in NP.
   How to prove? Provide a verifier.
2. VERTEX-COVER is NP-HARD.
   How to prove? Reduction from 3SAT to VERTEX-COVER, using clause and variable gadgets.

Theorem 7.44

VERTEX-COVER is NP-complete.

Proof:

1. VERTEX-COVER is in NP.
   How to prove? Provide a verifier.
2. VERTEX-COVER is NP-HARD.
   How to prove? Reduction from 3SAT to VERTEX-COVER, using clause and variable gadgets.
   $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

HAMPATH

HAMPATH $= \{\langle G, s, t\rangle | G$ is a directed graph with a Hamiltonian path from s to t $\}$.

A ***Hamiltonian path*** is a directed path that goes through each node in G exactly once.

HAMPATH

HAMPATH $= \{\langle G, s, t\rangle | G$ is a directed graph with a Hamiltonian path from s to t $\}$.

A ***Hamiltonian path*** is a directed path that goes through each node in G exactly once.

What does a certificate for HAMPATH look like?

Theorem 7.44

HAMPATH is NP-complete.

Theorem 7.44

HAMPATH is NP-complete.

Proof:

Theorem 7.44

HAMPATH is NP-complete.

Proof:

1. HAMPATH is in NP.
   How to prove?

Theorem 7.44

HAMPATH is NP-complete.

Proof:

1. HAMPATH is in NP.
   How to prove? Provide a verifier.

Theorem 7.44

HAMPATH is NP-complete.

Proof:

1. HAMPATH is in NP.
   How to prove? Provide a verifier.
2. HAMPATH is NP-HARD.
   How to prove?

Theorem 7.44

HAMPATH is NP-complete.

Proof:

1. HAMPATH is in NP.
   How to prove? Provide a verifier.
2. HAMPATH is NP-HARD.
   How to prove? Reduction from 3SAT to HAMPATH, using diamond variable gadgets and clause nodes.

Theorem 7.44

HAMPATH is NP-complete.

Proof:

1. HAMPATH is in NP.
   How to prove? Provide a verifier.
2. HAMPATH is NP-HARD.
   How to prove? Reduction from 3SAT to HAMPATH, using diamond variable gadgets and clause nodes.
   Diamond rows have 2 nodes per clause, with buffer node between, connected to clause node left-to-right loop, if positive literal in clause, right to left if negative literal in clause.

SAT is NP-hard

Reading: Sipser Theorem 7.37.

SAT Definition

SAT $= \{\langle\phi\rangle | \phi$ is a satisfiable Boolean formula over variables $x_1, x_2, ..., x_n\}$.

A $\in NP$ Definition

Let A $\in$ NP be any language in NP. Let N_{A} be a non-deterministic Turing machine that decides A. In other words, on input w, N_{A} will accept if $w \in$ A and reject if $w \notin$ A, in polynomial time n^k.

Reduction from A to SAT

$$\text{A} \leq_p \text{SAT}$$

Let $R =$ "On input $w, \langle N_{\text{A}} \rangle$:

1. Construct ϕ from w and N_{A}.
2. Output $\langle \phi \rangle$."

Functionality of Non-deterministic Turing Machines

- All branches process simultaneously.
- Each node in tree represented by a configuration.
- If machine is polynomial, tallest branch is at most $O(n^k)$ high.
- If machine is polynomial, largest configuration is at most $O(n^k)$ long.
- If any branch reaches an accepting configuration, the machine accepts.
- Path from initial configuration to accepting configuration is a list of consistent configurations.

Tableau of Configurations

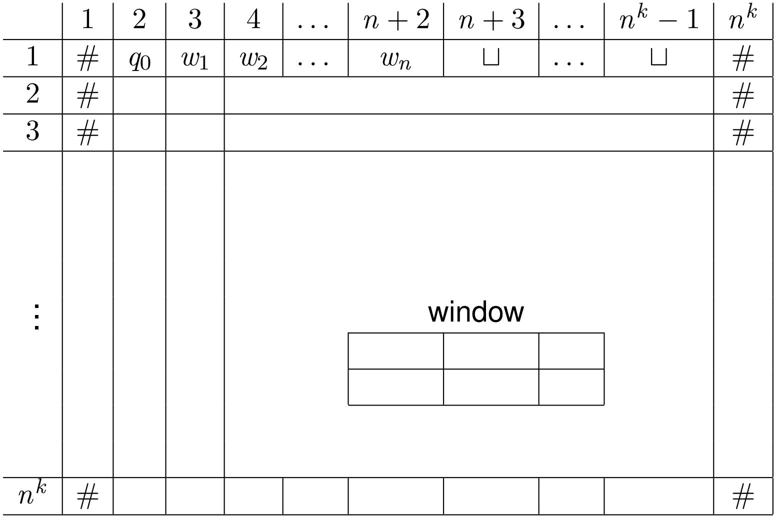


Tableau of Configurations

Properties of Tableau of Configurations

- Each row is a configuration of computation.
- Row 1 is the initial configuration.
- Row $i+1$ is one of the configurations that follows row i, according to the transition function.
- If any row contains q_{accept}, then the tableau is an accepting branch of the machine.
- Each cell in the tableau contains exactly one symbol $s \in C = Q \cup \Gamma \cup \{\#\}$.
- If there is an accepting tableau for N_{A}, w, then N_{A} accepts w.

Constructing ϕ

$\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$

ϕ_{cell}

Let $x_{i,j,s}$ for $1 \leq i, j \leq n^k$ and $s \in C$ be 1 if cell i, j of the tableau contains s and 0 otherwise.

ϕ_{cell}

Let $x_{i,j,s}$ for $1 \leq i, j \leq n^k$ and $s \in C$ be 1 if cell i, j of the tableau contains s and 0 otherwise.

Each cell must contain a symbol: $\bigvee_{s \in C} x_{i,j,s}$

ϕ_{cell}

Let $x_{i,j,s}$ for $1 \leq i, j \leq n^k$ and $s \in C$ be 1 if cell i, j of the tableau contains s and 0 otherwise.

Each cell must contain a symbol: $\bigvee_{s \in C} x_{i,j,s}$

Each cell must not contain two symbols: $\bigwedge_{s,t \in C; s \neq t} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})$

ϕ_{cell}

Let $x_{i,j,s}$ for $1 \leq i, j \leq n^k$ and $s \in C$ be 1 if cell i, j of the tableau contains s and 0 otherwise.

Each cell must contain a symbol: $\displaystyle\bigvee_{s \in C} x_{i,j,s}$

Each cell must not contain two symbols: $\displaystyle\bigwedge_{s,t \in C; s \neq t} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}})$

All cells must contain exactly one symbol:

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i,j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{s,t \in C; s \neq t} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

ϕ_{start}

We need to represent the first row of the tableau as a logic formula.

1	2	3	4	$\ldots$	$n+2$	$n+3$	$\ldots$	n^k-1	n^k
$\#$	q_0	w_1	w_2	$\ldots$	w_n	$\sqcup$	$\ldots$	$\sqcup$	$\#$

ϕ_{start}

We need to represent the first row of the tableau as a logic formula.

1	2	3	4	$\dots$	$n+2$	$n+3$	$\dots$	n^k-1	n^k
$\#$	q_0	w_1	w_2	$\dots$	w_n	$\sqcup$	$\dots$	$\sqcup$	$\#$

$$\begin{aligned}\phi_{\text{start}} = \quad & x_{1,1,\#} \wedge x_{1,2,q_0} \\ & \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \cdots \wedge x_{1,n+2,w_n} \\ & \wedge x_{1,n+3,\sqcup} \wedge \cdots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}\end{aligned}$$

ϕ_{accept}

If an accepting branch exists, then the accept state must be present somewhere in the tableau.

$$\phi_{\text{accept}} = \bigvee_{1 \le i,j \le n^k} x_{1,1,q_{accept}}$$

ϕ_{move}

For configuration $i + 1$ to be allowable, it needs to be legal to move from configuration i to configuration $i + 1$, according to the transition function of N_{A}.

ϕ_{move}

For configuration $i+1$ to be allowable, it needs to be legal to move from configuration i to configuration $i+1$, according to the transition function of N_{A}.

Changes between configurations i and $i+1$ will only occur in a small window around the location of the state. We call this location (i,j) as the middle-upper cell in the 2x3 window.

ϕ_{move}

For configuration $i+1$ to be allowable, it needs to be legal to move from configuration i to configuration $i+1$, according to the transition function of N_{A}.

Changes between configurations i and $i+1$ will only occur in a small window around the location of the state. We call this location (i,j) as the middle-upper cell in the 2x3 window.

$$\phi_{\text{move}} = \bigwedge_{1\leq i<n^k, 1<j<n^k} \text{(the window at } (i,j) \text{ is legal}$$

ϕ_{move}

For configuration $i + 1$ to be allowable, it needs to be legal to move from configuration i to configuration $i + 1$, according to the transition function of N_{A}.

Changes between configurations i and $i + 1$ will only occur in a small window around the location of the state. We call this location (i, j) as the middle-upper cell in the 2x3 window.

$$\phi_{\text{move}} = \bigwedge_{1 \leq i < n^k, 1 < j < n^k} \text{(the window at } (i, j) \text{ is legal}$$

$$\bigvee_{a_1, \ldots, a_6; legalwindow} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+$$

ϕ_{move}

For configuration $i+1$ to be allowable, it needs to be legal to move from configuration i to configuration $i+1$, according to the transition function of N_{A}.

Changes between configurations i and $i+1$ will only occur in a small window around the location of the state. We call this location (i,j) as the middle-upper cell in the 2x3 window.

$$\phi_{\text{move}} = \bigwedge_{1\leq i<n^k, 1<j<n^k} \text{(the window at } (i,j) \text{ is legal}$$

$$\bigvee_{a_1,\ldots,a_6; legalwindow} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+$$