Why Study the Theory of Computation?

Implementations come and go.

Chapter 1
IBM 7090 Programming in the 1950’s

ENTRY      SXA     4,RETURN
LDQ       X
FMP       A
FAD       B
XCA
FMP       X
FAD       C
STO      RESULT

RETURN     TRA     0
A        BSS    1
B        BSS    1
C        BSS    1
X        BSS    1
TEMP      BSS    1
STORE     BSS    1
END
Programming in the 1970’s (IBM 360)

//MYJOB JOB (COMPRESS),
'VOLKER BANDKE', CLASS=P, COND=(0, NE)
//BACKUP EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=* 
//SYSUT1 DD DISP=SHR, DSN=MY.IMPORTNT.PDS 
//SYSUT2 DD DISP=(, CATLG),
DSN=MY.IMPORTNT.PDS.BACKUP,
// UNIT=3350, VOL=SER=DISK01,
// DCB=MY.IMPORTNT.PDS,
SPACE=(CYL, (10, 10, 20))

//COMPRESS EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=* 
//MYPDS DD DISP=OLD, DSN=* .BACKUP.SYSUT1 
//SYSIN DD *
COPY INDD=MYPDS, OUTDD=MYPDS
//DELETE2 EXEC PGM=IEFBR14
//BACKPDS DD DISP=(OLD, DELETE, DELETE),
DSN=MY.IMPORTNT.PDS.BACKUP
Guruhood

\[(\Gamma / V) > (+/V) - \Gamma / V\]
**Disjunctive Normal Form for Queries**

(category = fruit and supplier = A)

<table>
<thead>
<tr>
<th>Category</th>
<th>Supplier</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruit</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(category = fruit or category = vegetable)

<table>
<thead>
<tr>
<th>Category</th>
<th>Supplier</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vegetable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Disjunctive Normal Form for Queries

(category = fruit or category = vegetable)
and
(supplier = A or supplier = B)

<table>
<thead>
<tr>
<th>Category</th>
<th>Supplier</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruit</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>vegetable</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>
Disjunctive Normal Form for Queries

(category = fruit or category = vegetable) and (supplier = A or supplier = B)

becomes

(category = fruit and supplier = A) or (category = fruit and supplier = B) or (category = vegetable and supplier = A) or (category = vegetable and supplier = B)

<table>
<thead>
<tr>
<th>Category</th>
<th>Supplier</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruit</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>fruit</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>vegetable</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>vegetable</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>
Applications of the Theory

• FSMs for parity checkers, vending machines, communication protocols, and building security devices.
• Interactive games as nondeterministic FSMs.
• Programming languages, compilers, and context-free grammars.
• Natural languages are mostly context-free. Speech understanding systems use probabilistic FSMs.
• Computational biology: DNA and proteins are strings.
• The undecidability of a simple security model.
• Artificial intelligence: the undecidability of first-order logic.
Languages and Strings

Chapter 2
Let's Look at Some Problems

```c
int alpha, beta;
alpha = 3;
beta = (2 + 5) / 10;
```

(1) **Lexical analysis**: Scan the program and break it up into variable names, numbers, etc.
(2) **Parsing**: Create a tree that corresponds to the sequence of operations that should be executed, e.g.,
```
+ 10
 / 
+  2
   5
```
(3) **Optimization**: Realize that we can skip the first assignment since the value is never used and that we can precompute the arithmetic expression, since it contains only constants.
(4) **Termination**: Decide whether the program is guaranteed to halt.
(5) **Interpretation**: Figure out what (if anything) useful it does.
A Framework for Analyzing Problems

We need a single framework in which we can analyze a very diverse set of problems.

The framework we will use is

Language Recognition

A language is a (possibly infinite) set of finite length strings over a finite alphabet.
Strings

A string is a finite sequence, possibly empty, of symbols drawn from some alphabet $\Sigma$.

- $\varepsilon$ is the empty string.
- $\Sigma^*$ is the set of all possible strings over an alphabet $\Sigma$.

<table>
<thead>
<tr>
<th>Alphabet name</th>
<th>Alphabet symbols</th>
<th>Example strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>The English alphabet</td>
<td>${a, b, c, \ldots, z}$</td>
<td>$\varepsilon, \text{aabbcg, aaaaa}$</td>
</tr>
<tr>
<td>The binary alphabet</td>
<td>${0, 1}$</td>
<td>$\varepsilon, 0, 001100$</td>
</tr>
<tr>
<td>A star alphabet</td>
<td>${★, ✪, ✫, ✯, ✡, ✰}$</td>
<td>$\varepsilon, ✪✪, ✪✯✯✰★✰$</td>
</tr>
<tr>
<td>A music alphabet</td>
<td>${w, h, q, e, x, r, ●}$</td>
<td>$\varepsilon, w \mid h h \mid hqq \mid$</td>
</tr>
</tbody>
</table>
Functions on Strings

**Counting:** \(|s|\) is the number of symbols in \(s\).

\[
|\varepsilon| = 0 \\
|1001101| = 7
\]

\(\#_c(s)\) is the number of times that \(c\) occurs in \(s\).

\[
\#_a(abbaaa) = 4.
\]
More Functions on Strings

**Concatenation:** $st$ is the *concatenation* of $s$ and $t$.

If $x = \text{good}$ and $y = \text{bye}$, then $xy = \text{goodbye}$.

Note that $|xy| = |x| + |y|$.

$\epsilon$ is the identity for concatenation of strings. So:

$$\forall x (x \epsilon = \epsilon x = x).$$

Concatenation is associative. So:

$$\forall s, t, w ((st)w = s(tw)).$$
More Functions on Strings

**Replication:** For each string $w$ and each natural number $i$, the string $w^i$ is:

\[
\begin{align*}
    w^0 &= \varepsilon \\
    w^{i+1} &= w^i w
\end{align*}
\]

Examples:

\[
\begin{align*}
    a^3 &= aaa \\
    (\text{bye})^2 &= \text{byebye} \\
    a^0b^3 &= bbb
\end{align*}
\]
More Functions on Strings

**Reverse**: For each string $w$, $w^R$ is defined as:

if $|w| = 0$ then $w^R = w = \varepsilon$

if $|w| \geq 1$ then:

$\exists a \in \Sigma \ (\exists u \in \Sigma^* \ (w = ua))$.

So define $w^R = a \ u^R$. 
Concatenation and Reverse of Strings

**Theorem:** If $w$ and $x$ are strings, then $(w \ x)^R = x^R \ w^R$.

Example:

$$(\text{nametag})^R = (\text{tag})^R \ (\text{name})^R = \text{gateman}$$
Concatenation and Reverse of Strings

Proof: By induction on $|x|:$

$|x| = 0$: Then $x = \varepsilon$, and $(wx)^R = (w \varepsilon)^R = (w)^R = \varepsilon w^R = \varepsilon^R w^R = x^R w^R.$

$\forall n \geq 0 ((|x| = n) \rightarrow ((w x)^R = x^R w^R)) \rightarrow (|x| = n + 1) \rightarrow ((w x)^R = x^R w^R))$: 

Consider any string $x$, where $|x| = n + 1$. Then $x = u a$ for some character $a$ and $|u| = n$. So:

\[
(w x)^R = (w (u a))^R \\
= (((w u) a)^R \\
= (a (w u))^R \\
= (a (u^R w^R))^R \\
= (a u^R) w^R \\
= (u a) w^R \\
= x^R w^R
\]

rewrite $x$ as $ua$

associativity of concatenation

definition of reversal

induction hypothesis

associativity of concatenation

definition of reversal

rewrite $ua$ as $x$
Relations on Strings

aaa is a substring of aaabbbbaaa

aaaaaa is not a substring of aaabbbbaaa

aaa is a proper substring of aaabbbbaaa

Every string is a substring of itself.

ε is a substring of every string.
The Prefix Relations

s is a prefix of t iff: \( \exists x \in \Sigma^* (t = sx) \).

s is a proper prefix of t iff: s is a prefix of t and s \( \neq t \).

Examples:

The prefixes of abba are: \( \varepsilon, a, ab, abb, abba \).
The proper prefixes of abba are: \( \varepsilon, a, ab, abb \).

Every string is a prefix of itself.

\( \varepsilon \) is a prefix of every string.
The Suffix Relations

s is a **suffix** of t iff: \( \exists x \in \Sigma^* \,(t = xs) \).

s is a **proper suffix** of t iff: s is a suffix of t and \( s \neq t \).

Examples:

The **suffixes** of abba are: \( \varepsilon, a, ba, bba, abba \).

The **proper suffixes** of abba are: \( \varepsilon, a, ba, bba \).

Every string is a suffix of itself.

\( \varepsilon \) is a suffix of every string.
A **language** is a (finite or infinite) set of strings over a finite alphabet $\Sigma$.

Examples: Let $\Sigma = \{a, b\}$

Some languages over $\Sigma$:
- $\emptyset$,
- $\{\varepsilon\}$,
- $\{a, b\}$,
- $\{\varepsilon, a, aa, aaa, aaaa, aaaaa\}$

The language $\Sigma^*$ contains an infinite number of strings, including: $\varepsilon, a, b, ab, ababaa$. 
Example Language Definitions

\[ L = \{ x \in \{a, b\}^* : \text{all } a \text{'s precede all } b \text{'s} \} \]

\[ \varepsilon, a, aa, aabbb, \text{ and } bb \text{ are in } L. \]

\[ aba, ba, \text{ and } abc \text{ are not in } L. \]

What about: \( \varepsilon, a, aa, \text{ and } bb? \)
Example Language Definitions

\[ L = \{ x : \exists y \in \{a, b\}^* : x = ya \} \]

Simple English description:
The Perils of Using English

$L = \{x#y: x, y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^* \text{ and, when } x \text{ and } y \text{ are viewed as the decimal representations of natural numbers, } square(x) = y\}.

Examples:

3#9, 12#144

3#8, 12, 12#12#12
More Example Language Definitions

\[ L = \{\} = \emptyset \]

\[ L = \{\varepsilon\} \]
English

$L = \{w: w \text{ is a sentence in English}\}.$

Examples:

Kerry hit the ball.

Colorless green ideas sleep furiously.

The window needs fixed.

Ball the Stacy hit blue.
A Halting Problem Language

$L = \{ w : w \text{ is a C program that halts on all inputs} \}.$

- Well specified.
- Can we decide what strings it contains?
Prefixes

What are the following languages:

\[ L = \{ w \in \{a, b\}^* : \text{no prefix of } w \text{ contains } b \} \]

\[ L = \{ w \in \{a, b\}^* : \text{no prefix of } w \text{ starts with } a \} \]

\[ L = \{ w \in \{a, b\}^* : \text{every prefix of } w \text{ starts with } a \} \]
Using Replication in a Language
Definition

\[ L = \{a^n : n \geq 0\} \]
Languages Are Sets

Computational definition:

• Generator (enumerator)

• Recognizer
Enumeration

Enumeration:

- Arbitrary order

- More useful: lexicographic order
  - Shortest first
  - Within a length, dictionary order

The lexicographic enumeration of:

- \( \{w \in \{a, b\}^* : |w| \text{ is even}\} : \)
How Large is a Language?

The smallest language over any $\Sigma$ is $\emptyset$, with cardinality 0.

The largest is $\Sigma^*$. How big is it?
How Large is a Language?

**Theorem:** If $\Sigma \neq \emptyset$ then $\Sigma^*$ is countably infinite.

**Proof:** The elements of $\Sigma^*$ can be lexicographically enumerated by the following procedure:

- Enumerate all strings of length 0, then length 1, then length 2, and so forth.
- Within the strings of a given length, enumerate them in dictionary order.

This enumeration is infinite since there is no longest string in $\Sigma^*$. Since there exists an infinite enumeration of $\Sigma^*$, it is countably infinite.
How Large is a Language?

So the smallest language has cardinality 0.

The largest is countably infinite.

So every language is either finite or countably infinite.
How Many Languages Are There?

**Theorem:** If $\Sigma \neq \emptyset$ then the set of languages over $\Sigma$ is uncountably infinite.

**Proof:** The set of languages defined on $\Sigma$ is $P(\Sigma^*)$. $\Sigma^*$ is countably infinite. If $S$ is a countably infinite set, $P(S)$ is uncountably infinite. So $P(\Sigma^*)$ is uncountably infinite.
Functions on Languages

• Set operations
  • Union
  • Intersection
  • Complement

• Language operations
  • Concatenation
  • Kleene star
Concatenation of Languages

If $L_1$ and $L_2$ are languages over $\Sigma$:

$$L_1L_2 = \{w \in \Sigma^* : \exists s \in L_1 (\exists t \in L_2 (w = st))\}$$

Examples:

$L_1 = \{\text{cat, dog}\}$
$L_2 = \{\text{apple, pear}\}$
$L_1 L_2 = \{\text{catapple, catpear, dogapple, dogpear}\}$

$L_1 = a^*$
$L_2 = b^*$
$L_1 L_2 =$
Concatenation of Languages

\{\varepsilon\} is the identity for concatenation:

\[ L\{\varepsilon\} = \{\varepsilon\}L = L \]

\emptyset is a zero for concatenation:

\[ L \emptyset = \emptyset L = \emptyset \]
Concatenating Languages Defined Using Variables

The scope of any variable used in an expression that invokes replication will be taken to be the entire expression.

\[ L_1 = \{a^n : n \geq 0\} \]
\[ L_2 = \{b^n : n \geq 0\} \]

\[ L_1 L_2 = \{a^n b^m : n, m \geq 0\} \]
\[ L_1 L_2 \neq \{a^n b^n : n \geq 0\} \]
Kleene Star

\( L^* = \{ \varepsilon \} \cup \{ w \in \Sigma^* : \exists k \geq 1 \) \\
(\exists w_1, w_2, \ldots w_k \in L \ (w = w_1 w_2 \ldots w_k) \}) \)

Example:

\( L = \{ \text{dog, cat, fish} \} \)
\( L^* = \{ \varepsilon, \text{dog, cat, fish, dogdog,} \) \\
\text{dogcat, fishcatfish,} \)
\( \text{fishdogdogfishcat, ...} \)
The $+$ Operator

$L^+ = L \cdot L^*$

$L^+ = L^* - \{\varepsilon\}$  iff  $\varepsilon \notin L$

$L^+$ is the closure of $L$ under concatenation.
Concatenation and Reverse of Languages

**Theorem:** \((L_1L_2)^R = L_2^R L_1^R\).

**Proof:**
\[
\forall x \ (\forall y \ ((xy)^R = y^Rx^R)) \quad \text{Theorem 2.1}
\]

\[
(L_1L_2)^R = \{(xy)^R : x \in L_1 \text{ and } y \in L_2\} \quad \text{Definition of concatenation of languages}
\]

\[
= \{y^Rx^R : x \in L_1 \text{ and } y \in L_2\} \quad \text{Lines 1 and 2}
\]

\[
= L_2^R L_1^R \quad \text{Definition of concatenation of languages}
\]
What About Meaning?

$A^n B^n = \{a^n b^n : n \geq 0\}$.

Do these strings mean anything?
Semantic Interpretation Functions

A semantic interpretation function assigns meanings to the strings of a language.

English:

I brogelled the yourtish.

The semantic interpretation function for English is mostly compositional.

He’s all thumbs.
Semantic Interpretation Functions

For formal languages:

- Programming languages
- Network protocol languages
- Database query languages
- HTML
- BNF

For other kinds of “natural” languages:

- DNA
Chapter 3
A decision problem is simply a problem for which the answer is yes or no (True or False). A decision procedure answers a decision problem.

Examples:

• Given an integer $n$, does $n$ have a pair of consecutive integers as factors?

• The language recognition problem: Given a language $L$ and a string $w$, is $w$ in $L$?

Our focus
The Power of Encoding

Everything is a string.

Problems that don’t look like decision problems can be recast into new problems that do look like that.
The Power of Encoding

Pattern matching on the web:

- Problem: Given a search string $w$ and a web document $d$, do they match? In other words, should a search engine, on input $w$, consider returning $d$?

- The language to be decided: $\{<w, d> : d \text{ is a candidate match for the query } w\}$
The Power of Encoding

Does a program always halt?

• Problem: Given a program $p$, written in some standard programming language, is $p$ guaranteed to halt on all inputs?

• The language to be decided:

$$\text{HP}_{\text{ALL}} = \{p : p \text{ halts on all inputs}\}$$
What If We’re Not Working with Strings?

Anything can be encoded as a string.

\(<X>\) is the string encoding of \(X\).
\(<X, Y>\) is the string encoding of the pair \(X, Y\).
Primality Testing

• Problem: Given a nonnegative integer $n$, is it prime?
• An instance of the problem: Is 9 prime?
• To encode the problem we need a way to encode each instance: We encode each nonnegative integer as a binary string.
• The language to be decided:
  \[ PRIMES = \{ w : w \text{ is the binary encoding of a prime number} \}. \]
Problem: Given an undirected graph $G$, is it connected?

Instance of the problem:

```
1           2            3
\downarrow  4           5
```

Encoding of the problem: Let $V$ be a set of binary numbers, one for each vertex in $G$. Then we construct $\langle G \rangle$ as follows:

- Write $|V|$ as a binary number,
- Write a list of edges,
- Separate all such binary numbers by "/".

```
101/1/10/10/11/1/100/10/101
```

The language to be decided: CONNECTED = \{ $w \in \{0, 1, /\}^*$ : $w = n_1/n_2/\ldots/n_i$, where each $n_i$ is a binary string and $w$ encodes a connected graph, as described above\}. 

The Power of Encoding
The Power of Encoding

• Protein sequence alignment:

• Problem: Given a protein fragment $f$ and a complete protein molecule $p$, could $f$ be a fragment from $p$?

• Encoding of the problem: Represent each protein molecule or fragment as a sequence of amino acid residues. Assign a letter to each of the 20 possible amino acids. So a protein fragment might be represented as AGHTYWDNR.

• The language to be decided: $\{<f, p> : f \text{ could be a fragment from } p\}$. 
Casting multiplication as decision:

• Problem: Given two nonnegative integers, compute their product.

• Encoding of the problem: Transform computing into verification.

• The language to be decided:

\[ L = \{ w \text{ of the form: } <\text{integer}_1> \times <\text{integer}_2> = <\text{integer}_3>, \text{ where: } <\text{integer}_n> \text{ is any well formed integer, and } \text{integer}_3 = \text{integer}_1 \times \text{integer}_2 \} \]

12 \times 9 = 108
12 = 12
12 \times 8 = 108
Turning Problems Into Decision Problems

Casting sorting as decision:

• Problem: Given a list of integers, sort it.

• Encoding of the problem: Transform the sorting problem into one of examining a pair of lists.

• The language to be decided:

\[ L = \{ w_1 \neq w_2 : \exists n \geq 1 \text{ (w_1 is of the form <int_1, int_2, … int}_n>, w_2 \text{ is of the form <int}_1, \text{ int}_2, \text{ … } \text{ int}_n>, \text{ and w}_2 \text{ contains the same objects as } w_1 \text{ and w}_2 \text{ is sorted)}\} \]

Examples:

1, 5, 3, 9, 6 # 1, 3, 5, 6, 9 \in L
1, 5, 3, 9, 6 # 1, 2, 3, 4, 5, 6, 7 \notin L
Turning Problems Into Decision Problems

Casting database querying as decision:

• Problem: Given a database and a query, execute the query.

• Encoding of the problem: Transform the query execution problem into evaluating a reply for correctness.

• The language to be decided:

$L = \{ d \# q \# a: \quad$

\hspace{1cm} $d$ is an encoding of a database,

\hspace{1cm} $q$ is a string representing a query, and

\hspace{1cm} $a$ is the correct result of applying $q$ to $d}\}$

Example:

$(\text{name, age, phone}), (\text{John, 23, 567-1234})$

$(\text{Mary, 24, 234-9876})#(\text{select name age=23})#$

$(\text{John}) \in L$
The Traditional Problems and their Language Formulations are Equivalent

By equivalent we mean that either problem can be reduced to the other.

If we have a machine to solve one, we can use it to build a machine to do the other using just the starting machine and other functions that can be built using a machine of equal or lesser power.
An Example

Consider the multiplication example:

\[ L = \{ w \text{ of the form:} \]
\[ <integer_1> \times <integer_2> = <integer_3>, \text{ where:} \]
\[ <integer_n> \text{ is any well formed integer, and} \]
\[ integer_3 = integer_1 \times integer_2 \}\n
Given a multiplication machine, we can build the language recognition machine:

Given the language recognition machine, we can build a multiplication machine:
Languages and Machines
Finite State Machines

An FSM to accept $a^*b^*$:

An FSM to accept $A^nB^n = \{a^m b^n : n \geq 0\}$
Pushdown Automata

A PDA to accept $A^nB^n = \{a^n b^n : n \geq 0\}$

Example: $aaabb$

Stack:
Another Example

Bal, the language of balanced parentheses
Trying Another PDA

A PDA to accept strings of the form:

$$A^nB^nC^n = \{a^nb^nC^n : n \geq 0\}$$
Turing Machines

A Turing Machine to accept $A^nB^nC^n$:
A Turing machine to accept the language:

\{p: p \text{ is a Java program that halts on input } 0\}
Rule of Least Power: “Use the least powerful language suitable for expressing information, constraints or programs on the World Wide Web.”
Grammars, Languages, and Machines

Grammar \( L \) accepts Language

Language

Accepts

Machine
A Tractability Hierarchy

- \( P \)
- \( NP \)
- \( \text{PSPACE} \)

\( P \subseteq NP \subseteq \text{PSPACE} \)
Computation

Chapter 4
Three Computational Issues

• Decision procedures
• Nondeterminism
• Functions on functions and programs
Decision Procedures

Given two strings $s$ and $t$, does $s$ occur anywhere as a substring of $t$?
Decision Procedures
Fermat Numbers

\[ F_n = 2^{2^n} + 1, \quad n \geq 0 \]

\[ F_0 = 3, \quad F_1 = 5, \quad F_2 = 17, \quad F_3 = 257, \quad F_4 = 65,537, \]
\[ F_5 = 4,294,967,297 \]

• Are there any prime Fermat numbers less than 1,000,000?

• Are there any prime Fermat numbers greater than 1,000,000?
Decision Procedures

Given a Java program $P$, is there some input string on which $P$ halts?

Given a Java program $P$, and a value $v$, is $v$ the shortest input string on which $P$ halts?
Nondeterminism

1. choose (action 1;
   action 2;
   ...
   action \( n \) )

2. choose\( (x \text{ from } S: P(x)) \)
Nondeterminism

Given two strings $s$ and $t$, does $s$ occur anywhere as a substring of $t$?
Nondeterminism

\[
\text{trip-plan}(\text{start, finish}) = \\
\quad \text{return (choose(} \\
\quad \quad \text{fly-major-airline-and-rent-car}(\text{start, finish});; \\
\quad \quad \text{fly-regional-airline-and-rent-car}(\text{start, finish});; \\
\quad \quad \text{take-train-and-use-public-transportation} \\
\quad \quad \quad (\text{start, finish});; \\
\quad \quad \text{drive}(\text{start, finish}) \quad ))
\]
Nondeterminism

<table>
<thead>
<tr>
<th>5</th>
<th>2</th>
<th>15</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>8</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>
The 15-Puzzle

solve-15(position-list) =
   /* Explore moves available from the last board configuration to have
    been generated.
    current = last(position-list);
    if current = solution then return (position-list);
    /* Assume that successors(current) returns the set of configurations
     that can be generated by one legal move from current. No other
     condition needs to be checked, so choose simply picks one.
     append(position-list, choose x from successors(current): True);
    /* Recursively call solve-15 to continue searching from the new
     board configuration.
    return(solve-15(position-list));
Given a graph $G$, is there a Hamiltonian circuit (a closed loop through a graph that visits each node exactly once) through $G$?

```plaintext
findit(G, n);
    start := n
    Return(findHam(G, n))
```

```plaintext
findHam(G, n);
    Mark n
    If there are any unmarked successors of $n$ do
        Return(choose ($m$ from successors of $n$ :
                findHam(G, m)))
    Else if all elements of $G$ are marked and $start \in$ successors of $n$
        then return (True)
        else return (False)
```
Nondeterministically Deciding SAT

\[ \text{decideSAT}(w: \text{Boolean wff}) = \]
If there are no predicate symbols in \( w \) then:
   Simplify \( w \) until it is either \text{True} or \text{False}.
   Return \( w \).
Else:
   Find \( P \), the first predicate symbol in \( w \).
   /* Let \( w/P/x \) mean the wff \( w \) with every instance of \( P \)
      replaced by \( x \).
   Return \( \text{choose (decideSAT}(w/P/\text{True});; \text{decideSAT}(w/P/\text{False}))) \).
Implementing Nondeterminism

Before the first choice `choose` makes

- First call to `choose`
  - Choice 1
  - Second call to `choose`
    - Choice 1
    - Second call to `choose`
      - Choice 1
  - Choice 2
  - Second call to `choose`
    - Choice 2
      - Second call to `choose`
        - Choice 2
The SAT Example
Nondeterminism in Finite State Machines
Functions on Languages

$maxstring(L) = \{w \in L: \forall z \in \Sigma^* \ (z \neq \varepsilon \rightarrow wz \notin L)\}.$

Examples:

- $maxstring(A^nB^n)$
- $maxstring(a^*)$

Let INF be the set of infinite languages. Let FIN be the set of finite languages.

Are the classes FIN and INF closed under $maxstring$?
Functions on Languages

\( \text{chop}(L) = \{ w : \exists x \in L \ (x = x_1cx_2, \ x_1 \in \Sigma_L^*, \ x_2 \in \Sigma_L^*, \ c \in \Sigma_L, \ |x_1| = |x_2|, \ \text{and} \ w = x_1x_2) \}. \)

What is \( \text{chop}(A^nB^n) \)?

What is \( \text{chop}(A^nB^nC^n) \)?

Are FIN and INF closed under \( \text{chop} \)?
Functions on Languages

\[
\text{firstchars}(L) = \\
\{ w : \exists y \in L \ (y = cx \land c \in \Sigma_L \land x \in \Sigma_L^* \land w \in c^*) \}. 
\]

What is \( \text{firstchars}(A^nB^n) \)?

What is \( \text{firstchars}(\{a, b\}^*) \)?

Are FIN and INF closed under \( \text{firstchars} \)?
Representing Languages as Machines

Compute union using descriptions like:

- \( \{ w \in \{a, b\}^* : w \text{ has odd length} \} \)
- \( \{ w \in \{a, b\}^* : \text{all } a \text{'s in } w \text{ precede all } b \text{'s} \} \)

Compute union using descriptions like: