CSE 311 Foundations of Computing I
Autumn 2011
Lecture 29
Course Summary

About the course

- From the CSE catalog:
  - CSE 311 Foundations of Computing I (4)
    Examines fundamentals of logic, set theory, induction, and algebraic structures with applications to computing; finite state machines; and limits of computability. Prerequisite: CSE 143; either MATH 126 or MATH 136.
- What this course is about:
  - Foundational structures for the practice of computer science and engineering

Propositional Logic

- Statements with truth values
  - The Washington State flag is red
  - It snowed in Whistler, BC on January 4, 2011.
  - Rick Perry won the Iowa straw poll
  - Space aliens landed in Roswell, New Mexico
  - If \( n \) is an integer greater than two, then the equation \( a^n + b^n = c^n \) has no solutions in non-zero integers \( a, b, \) and \( c \).
- Propositional variables: \( p, q, r, s, \ldots \)
- Truth values: \( T \) for true, \( F \) for false
- Compound propositions

Negation (not) \( \neg p \)
Conjunction (and) \( p \land q \)
Disjunction (or) \( p \lor q \)
Exclusive or \( p \oplus q \)
Implication \( p \rightarrow q \)
Biconditional \( p \iff q \)

Logical equivalence

- Terminology: A compound proposition is a
  - Tautology if it is always true
  - Contradiction if it is always false
  - Contingency if it can be either true or false

\[ p \lor \neg p \]
\[ p \land p \]
\[ (p \rightarrow q) \land p \]
\[ (p \land q) \lor (p \land \neg q) \lor (\neg p \land q) \lor (\neg p \land \neg q) \]
Logical Equivalence

- $p$ and $q$ are **logically equivalent** iff $p \leftrightarrow q$ is a tautology
- The notation $p \equiv q$ denotes $p$ and $q$ are logically equivalent

- De Morgan's Laws:
  - $\neg (p \land q) \equiv \neg p \lor \neg q$
  - $\neg (p \lor q) \equiv \neg p \land \neg q$

Digital Circuits

- Computing with logic
  - $T$ corresponds to 1 or "high" voltage
  - $F$ corresponds to 0 or "low" voltage

- Gates
  - Take inputs and produce outputs
  - Several kinds of gates
  - Correspond to propositional connectives
    - Only symmetric ones (order of inputs irrelevant)

Combinational Logic Circuits

- Computing with logic
- Gates
- Functions
- De Morgan's Laws

A quick combinational logic example

- Calendar subsystem: number of days in a month (to control watch display)
- used in controlling the display of a wrist-watch LCD screen
- inputs: month, leap year flag
- outputs: number of days

Implementation as a combinational digital system

- Encoding:
  - how many bits for each input/output?
  - binary number for month
  - four wires for 28, 29, 30, and 31

<table>
<thead>
<tr>
<th>month</th>
<th>leap</th>
<th>d28</th>
<th>d29</th>
<th>d30</th>
<th>d31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>-</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Truth-table to logic to switches to gates
- $d28 = 1$ when month=0010 and leap=0
- $d28 = m8\cdot m4 \cdot m2 \cdot m1 \cdot \text{leap}$
- $d31 = 1$ when month=0001 or month=0011 or ... month=1100
- $d31 = (m8 \cdot m4 \cdot m2 \cdot m1) + (m8 \cdot m4 \cdot m2 \cdot m1) + ...$
- $d31 = (m8 \cdot m4 \cdot m2 \cdot m1)$
- $d31 = \text{can we simplify more?}$

Combination example (cont’d)
Combinational example (cont'd)

\[
d_{28} = m_8' \cdot m_4' \cdot m_2 \cdot m_1' \cdot \text{leap}'
\]

\[
d_{29} = m_8' \cdot m_4' \cdot m_2 \cdot m_1' \cdot \text{leap}
\]

\[
d_{30} = (m_8' \cdot m_4' \cdot m_2' \cdot m_1' + m_8' \cdot m_4' \cdot m_2' \cdot m_1') + (m_8 \cdot m_4' \cdot m_2' \cdot m_1' + m_8 \cdot m_4' \cdot m_2' \cdot m_1')
\]

\[
d_{31} = (m_8' \cdot m_4' \cdot m_2' \cdot m_1' + m_8' \cdot m_4' \cdot m_2' \cdot m_1') + (m_8' \cdot m_4' \cdot m_2' \cdot m_1' + m_8' \cdot m_4' \cdot m_2' \cdot m_1') + (m_8 \cdot m_4' \cdot m_2' \cdot m_1') + (m_8 \cdot m_4' \cdot m_2' \cdot m_1')
\]

A simple example: 1-bit binary adder

- Inputs: A, B, Carry-in
- Outputs: Sum, Carry-out

Mapping truth tables to logic gates

- Given a truth table:
  1. Write the Boolean expression
  2. Minimize the Boolean expression
  3. Draw as gates
  4. Map to available gates

Predicate Calculus

- Predicate or Propositional Function
  - A function that returns a truth value
  - "x is a cat"
  - "student x has taken course y"
  - "x > y"
  - \( \forall x \ P(x) \) : P(x) is true for every x in the domain
  - \( \exists x \ P(x) \) : There is an x in the domain for which P(x) is true
**Statements with quantifiers**

- \( \forall x (\text{Even}(x) \lor \text{Odd}(x)) \)
- \( \exists x (\text{Even}(x) \land \text{Prime}(x)) \)
- \( \forall x \exists y (\text{Greater}(y, x) \land \text{Prime}(y)) \)
- \( \forall x (\text{Prime}(x) \rightarrow (\text{Equal}(x, 2) \lor \text{Odd}(x))) \)
- \( \exists x \exists y (\text{Equal}(x, y + 2) \land \text{Prime}(x) \land \text{Prime}(y)) \)

**Proofs**

- Start with hypotheses and facts
- Use rules of inference to extend set of facts
- Result is proved when it is included in the set

**Simple Propositional Inference Rules**

- Excluded middle
  \[ p \lor \neg p \]
- Two inference rules per binary connective one to eliminate it, one to introduce it.
  \[ 
  \begin{align*}
  p \land q & \vdash p, q \\
  p, q & \vdash p \land q \\
  p \lor q \lor \neg p & \vdash \neg p \\
  p, \neg p & \vdash q \\
  p, p \rightarrow q & \vdash q \\
  p \rightarrow q & \vdash p \rightarrow q \\
  \end{align*}
  
  Direct Proof Rule
  \]

**Inference Rules for Quantifiers**

- \( \forall x P(x) \)
  \[ 
  \begin{align*}
  P(c) & \text{ for some } c \\
  & \vdash \exists x P(x) \\
  \forall x P(x) & \vdash P(a) \text{ for any } a \\
  \end{align*}
  
  “Let a be anything” ...
  \]

- \( \exists x P(x) \)
  \[ 
  \begin{align*}
  \forall x P(x) & \vdash \exists x P(x) \\
  \exists x P(x) & \vdash P(c) \text{ for some special } c \\
  \end{align*}
  
  \]

**Even and Odd**

- \( \exists y \ (x = 2y) \)
- \( \text{Odd}(x) = \exists y \ (x = 2y + 1) \)

**Characteristic vectors**

- Let \( U = \{1, \ldots, 10\} \), represent the set \( \{1,3,4,8,9\} \) with \( 1011000110 \)
- Bit operations:
  \[ \begin{align*}
  0110110100 \lor 0011010110 & = 0111110110 \\
  \end{align*} \]

- \( \text{ls} -l \)
  \[ 
  \text{drwxr-xr-x ... Documents/} \\
  \text{-rw-r--r-- ... file1} \\
  \]
### One-time pad

- Alice and Bob privately share random n-bit vector K
  - Eve does not know K
- Later, Alice has n-bit message m to send to Bob
  - Alice computes $C = m \oplus K$
  - Alice sends C to Bob
  - Bob computes $m = C \oplus K$ which is $(m \oplus K) \oplus K$
- Eve cannot figure out m from C unless she can guess K

### Arithmetic mod 7

- $a +_7 b = (a + b) \mod 7$
- $a \times_7 b = (a \times b) \mod 7$

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

### Division Theorem

Let $a$ be an integer and $d$ a positive integer. Then there are unique integers $q$ and $r$, with $0 \leq r < d$, such that $a = dq + r$.

$q = a \div d \quad r = a \mod d$

### Modular Arithmetic

Let $a$ and $b$ be integers, and $m$ be a positive integer. We say $a$ is congruent to $b$ modulo $m$ if $m$ divides $a - b$. We use the notation $a \equiv b \pmod{m}$ to indicate that $a$ is congruent to $b$ modulo $m$.

Let $a$ and $b$ be integers, and let $m$ be a positive integer. Then $a \equiv b \pmod{m}$ if and only if $a \mod m = b \mod m$.

Let $m$ be a positive integer. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then
- $a + c \equiv b + d \pmod{m}$
- $ac \equiv bd \pmod{m}$

Let $a$ and $b$ be integers, and let $m$ be a positive integer. Then $a \equiv b \pmod{m}$ if and only if $a \mod m = b \mod m$.

### Integer representation

**Signed integer representation**

Suppose $-2^{n-1} < x < 2^{n-1}$
- First bit as the sign, $n-1$ bits for the value

<table>
<thead>
<tr>
<th>Value</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>0110 0011</td>
</tr>
<tr>
<td>-18</td>
<td>1001 0010</td>
</tr>
</tbody>
</table>

**Two’s complement representation**

Suppose $0 \leq x < 2^n - 1$.
- $x$ is represented by the binary representation of $x$
- $-x$ is represented by the binary representation of $2^n - x$

<table>
<thead>
<tr>
<th>Value</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>0110 0011</td>
</tr>
<tr>
<td>-18</td>
<td>1110 1110</td>
</tr>
</tbody>
</table>

### Hashing

- Map values from a large domain, 0…M-1 in a much smaller domain, 0…n-1
- Index lookup
- Test for equality
- Hash(x) = $x \mod p$
  - (or $\text{Hash}(x) = (ax + b) \mod p$
- Often want the hash function to depend on all of the bits of the data
  - Collision management
Modular Exponentiation

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 2 & 3 & 4 & 5 & 6 \\
2 & 2 & 4 & 1 & 2 & 4 & 1 \\
3 & 3 & 6 & 2 & 5 & 1 & 4 \\
4 & 4 & 1 & 5 & 2 & 6 & 3 \\
5 & 5 & 3 & 1 & 6 & 4 & 2 \\
6 & 6 & 5 & 4 & 3 & 2 & 1 \\
\end{array}
\]

Arithmetic mod 7

Fast exponentiation
Repeated Squaring

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Primality

An integer \( p \) greater than 1 is called **prime** if the only positive factors of \( p \) are 1 and \( p \).

A positive integer that is greater than 1 and is not prime is called **composite**.

**Fundamental Theorem of Arithmetic**: Every positive integer greater than 1 has a unique prime factorization.

GCD, LCM and Factoring

\[
a = 2^3 \cdot 3^1 \cdot 5^2 \cdot 7^1 = 46,200
\]

\[
b = 2^1 \cdot 3^2 \cdot 5^3 \cdot 7^1 \cdot 13^1 = 204,750
\]

\[
\text{GCD}(a, b) = 2^{\min(3,1)} \cdot 3^{\min(1,2)} \cdot 5^{\min(2,3)} \cdot 7^{\min(1,1)} \cdot 13^{\min(0,1)}
\]

\[
\text{LCM}(a, b) = 2^{\max(3,1)} \cdot 3^{\max(1,2)} \cdot 5^{\max(2,3)} \cdot 7^{\max(1,1)} \cdot 13^{\max(0,1)}
\]

Euclid’s Algorithm

- \( \text{GCD}(x, y) = \text{GCD}(y, x \mod y) \)

```
int \text{GCD}(\text{int } a, \text{int } b) // a >= b, b > 0 */
int tmp;
int x = a;
int y = b;
while (y > 0){
    tmp = x % y;
    x = y;
    y = tmp;
}
return x;
```

Multiplicative Inverse mod m

Suppose \( \text{GCD}(a, m) = 1 \)

By Bézoit’s Theorem, there exist integers \( s \) and \( t \) such that \( sa + tm = 1 \).

\( s \) is the multiplicative inverse of \( a \):

\[
1 = (sa + tm) \mod m = sa \mod m
\]
Induction proofs

1. Prove $P(0)$
2. Let $k$ be an arbitrary integer $\geq 0$
3. Assume that $P(k)$ is true
4. ...
5. Prove $P(k+1)$ is true

$P(0)$
$\forall k (P(k) \rightarrow P(k+1))$
$\therefore \forall n P(n)$

Strong Induction

$P(0)$
$\forall k ((P(0) \land P(1) \land P(2) \land \ldots \land P(k)) \rightarrow P(k+1))$
$\therefore \forall n P(n)$

Recursive definitions of functions

- $F(0) = 0$; $F(n + 1) = F(n) + 1$
- $G(0) = 1$; $G(n + 1) = 2 \times G(n)$
- $0! = 1$; $(n+1)! = (n+1) \times n!$
- $f_0 = 0$; $f_1 = 1$; $f_n = f_{n-1} + f_{n-2}$

Strings

- The set $\Sigma^*$ of strings over the alphabet $\Sigma$ is defined
  - Basis: $\lambda \in S$. ($\lambda$ is the empty string)
  - Recursive: if $w \in \Sigma^*$, $x \in \Sigma$, then $wx \in \Sigma^*$

- Palindromes: strings that are the same backwards and forwards.
  - Basis: $\lambda$ is a palindrome and any $a \in \Sigma$ is a palindrome
  - If $p$ is a palindrome then $apa$ is a palindrome for every $a \in \Sigma$

Function definitions on recursively defined sets

- $\text{Len}(\lambda) = 0$;
- $\text{Len}(wx) = 1 + \text{Len}(w)$; for $w \in \Sigma^*$, $x \in \Sigma$

- $\text{Concat}(w, \lambda) = w$ for $w \in \Sigma^*$
- $\text{Concat}(w_1, w_2) = \text{Concat}(w_1, w_2,x)$ for $w_1, w_2 \in \Sigma^*$, $x \in \Sigma$

- Prove: $\text{Len}(\text{Concat}(x,y)) = \text{Len}(x) + \text{Len}(y)$ for all strings $x$ and $y$

Rooted Binary trees

- Basis: $\bullet$ is a rooted binary tree
- Recursive Step: If $T_1$ and $T_2$ are rooted binary trees then so is:

  $T_1$

  $\bullet$

  $T_2$

  $\bullet$
Functions defined on rooted binary trees

• \(\text{size}(\bullet) = 1\)

• \(\text{size}(\text{rooted binary tree}) = 1 + \text{size}(T_1) + \text{size}(T_2)\)

• \(\text{height}(\bullet) = 0\)

• \(\text{height}(\text{rooted binary tree}) = 1 + \max\{\text{height}(T_1), \text{height}(T_2)\}\)

Prove:
For every rooted binary tree \(T\), \(\text{size}(T) \leq 2^{\text{height}(T) + 1} - 1\)

Regular Expressions over \(\Sigma\)

• Each is a "pattern" that specifies a set of strings

• Basis:
  - \(\emptyset\), \(\lambda\) are regular expressions
  - \(a\) is a regular expression for any \(a \in \Sigma\)

• Recursive step:
  - If \(A\) and \(B\) are regular expressions then so are:
    - \((A \cup B)\)
    - \((AB)\)
    - \(A^*\)

Regular Expressions

- \(0^*\)
- \(0^*1^*\)
- \((0 \cup 1)^*\)
- \((0^*1^*)^*\)
- \((0 \cup 1)^*\ 0110 (0 \cup 1)^*\)
- \((0 \cup 1)^* (0110 \cup 100)(0 \cup 1)^*\)

Context-Free Grammars

- Example: \(S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \lambda\)

- Example: \(S \rightarrow 0S \mid S1 \mid \lambda\)

Sample Context-Free Grammars

- Grammar for \(\{0^n1^n : n \geq 0\}\) all strings with same # of 0's and 1's with all 0's before 1's.

- Example: \(S \rightarrow (S) \mid SS \mid \lambda\)

Building in Precedence in Simple Arithmetic Expressions

- \(E\) – expression (start symbol)
- \(T\) – term \(F\) – factor \(I\) – identifier \(N\) – number
- \(E \rightarrow T \mid E + T\)
- \(T \rightarrow F \mid F * T\)
- \(F \rightarrow (E) \mid I \mid N\)
- \(I \rightarrow x \mid y \mid z\)
- \(N \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\)
BNF for C

Definition of Relations

Let A and B be sets, 
A binary relation from A to B is a subset of A × B 

Let A be a set, 
A binary relation on A is a subset of A × A 

Let R be a relation on A 

R is reflexive iff (a, a) ∈ R for every a ∈ A

R is symmetric iff (a, b) ∈ R implies (b, a) ∈ R

R is antisymmetric iff (a, b) ∈ R and a ≠ b implies (b, a) ∈ R

R is transitive iff (a, b) ∈ R and (b, c) ∈ R implies (a, c) ∈ R

Combining Relations

Let R be a relation from A to B 
Let S be a relation from B to C 
The composite of R and S, S ◦ R is the relation from A to C defined 

S ◦ R = {(a, c) | ∃ b such that (a, b) ∈ R and (b, c) ∈ S}

Relations

(a, b) ∈ Parent: b is a parent of a 
(a, b) ∈ Sister: b is a sister of a 
Aunt = Sister ∘ Parent 
Grandparent = Parent ∘ Parent 

R^2 = R ◦ R = {(a, c) | ∃ b such that (a, b) ∈ R and (b, c) ∈ R}

R^0 = {(a, a) | a ∈ A}
R^1 = R
R^n+1 = R^n ◦ R

n-ary relations

Let A_1, A_2, ..., A_n be sets. An n-ary relation on these sets is a subset of A_1 × A_2 × ... × A_n.
Matrix representation for relations

Relation $R$ on $A = \{a_1, \ldots, a_p\}$

$$ m_{ij} = \begin{cases} 1 & \text{if} \ (a_i, a_j) \in R, \\ 0 & \text{if} \ (a_i, a_j) \notin R. \end{cases} $$

$$ \{(1,1), (1,2), (1,4), (2,1), (2,3), (3,2), (3,3), (4,2), (4,3)\} $$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Representation of relations

Directed Graph Representation (Digraph)

$$ \{(a, b), (a, a), (b, a), (c, a), (c, d), (c, e), (d, e)\} $$

Paths in relations

Let $R$ be a relation on a set $A$. There is a path of length $n$ from $a$ to $b$ if and only if $(a, b) \in R^n$. 

(a, b) is in the transitive-reflexive closure of $R$ if and only if there is a path from $a$ to $b$. (Note: by definition, there is a path of length 0 from $a$ to $a$.)

Finite state machines

States

Transitions on inputs

Start state and finals states

The language recognized by a machine is the set of strings that reach a final state

Accepts strings with an odd number of 1’s and an odd number of 0’s

Accept strings with a 1 three positions from the end
Product construction

– Combining FSMs to check two properties at once
• New states record states of both FSMs

State machines with output

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>L</td>
</tr>
<tr>
<td>S_1</td>
<td>S_0</td>
</tr>
<tr>
<td>S_2</td>
<td>S_1</td>
</tr>
<tr>
<td>S_3</td>
<td>S_2</td>
</tr>
<tr>
<td>S_4</td>
<td>S_3</td>
</tr>
</tbody>
</table>

"Tug-of-war"

Vending Machine

Enter 15 cents in dimes or nickels
Press S or B for a candy bar

Vending Machine, Final Version

Vending Machine, Buggy Version

State minimization

Finite State Machines with output at states
Another way to look at DFAs

Definition: The label of a path in a DFA is the concatenation of all the labels on its edges in order.

Lemma: x is in the language recognized by a DFA iff x labels a path from the start state to some final state.

Nondeterministic Finite Automaton (NFA)

- Graph with start state, final states, edges labeled by symbols (like DFA) but
  - Not required to have exactly 1 edge out of each state labeled by each symbol - can have 0 or >1
  - Also can have edges labeled by empty string \( \lambda \)
- Definition: x is in the language recognized by an NFA iff x labels a path from the start state to some final state.

Building a NFA from a regular expression

\[(01 \cup 1)^*0\]

The set B of binary palindromes cannot be recognized by any DFA

Consider the infinite set of strings

\[S = \{\lambda, 0, 00, 000, 0000, \ldots\}\]

Claim: No two strings in S can end at the same state of any DFA for B, so no such DFA can exist.

Proof: Suppose \( n \neq m \) and \( 0^n \) and \( 0^m \) end at the same state p.
Since \( 0^n10^n \) is in B, following \( 10^n \) after state p must lead to a final state.
But then the DFA would accept \( 0^n10^n \) which is a contradiction.
Cardinality

• A set S is countable iff we can write it as $S = \{s_1, s_2, s_3, \ldots\}$ indexed by $\mathbb{N}$
• Set of integers is countable
  – $\{0, -1, 1, 2, -2, 3, -3, 4, \ldots\}$
• Set of rationals is countable
  – “dovetailing”
• $\Sigma^*$ is countable
  – $\{0, 1\}^* = \{0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, \ldots\}$
• Set of all (Java) programs is countable

The real numbers are not countable

• “diagonalization”

General models of computation

Control structures with infinite storage
Many models
Turing machines
Functional
Recursion
Java programs

Church-Turing Thesis
Any reasonable model of computation that includes all possible algorithms is equivalent in power to a Turing machine

Halting Problem

• Given: the code of a program $P$ and an input $x$ for $P$, i.e. given $(<P>, x)$
• Output: 1 if $P$ halts on input $x$
  0 if $P$ does not halt on input $x$

Theorem (Turing): There is no program that solves the halting problem
“The halting problem is undecidable”

Suppose $H(p, x)$ solves the Halting problem

Function $D(x)$:
if $H(p, x) = 1$ then
  while true; /* loop forever */
else
  no-op; /* do nothing and halt */
endif

$D$ halts on input $<D>$
$\iff H$ outputs 1 on input $(<D>, <D>)$
  [since $H$ solves the halting problem and so $H(<D>, x)$ outputs 1 iff $D$ halts on input $x$]
$D$ runs forever on input $<D>$
  [since $D$ goes into an infinite loop on $x$ iff $H(x, x) = 1$]
Does a program have a divide by 0 error?

**Input:** A program \(<P>\) and an input string \(x\)

**Output:** 1 if \(P\) has a divide by 0 error on input \(x\)
0 otherwise

Claim: The divide by zero problem is undecidable

---

**Program equivalence**

**Input:** the codes of two programs, \(<P>\) and \(<Q>\)

**Output:** 1 if \(P\) produces the same output as \(Q\) does on every input
0 otherwise

Claim: The equivalent program problem is undecidable

---

That’s all folks!

---

**Teaching evaluation**

- Please answer the questions on both sides of the form. This includes the ABET questions on the back

---