Announcements

• Hand in Homework 8 now
• Pick up all old homework now
• Review session
  – Sunday, June 9, 4 pm, EEB 125
  – List of Final Exam Topics and sampling of some
typical kinds of exam questions on the web
  – Bring your questions to the review session!
• Final exam
  – Monday, June 10, 2:30-4:20 pm, MGH 389

What makes the Halting Problem and related problems hard?

• Figuring out in some finite time something about what might happen in the infinite # of steps that
  a general computation might take
  – e.g. having $H(<P>,x)$ output 0 when $P$ doesn’t halt on $x$

• The following program does exist and won’t generate any contradiction
  – Function $V(x)$:
    • if $U(x,x)$ halts then
      – while (true); /* loop forever */
    • else
      – no-op; /* do nothing and halt */
    • endif

The “Always Halting” problem

Suppose we had a TM $A$ for the Always Halting problem

1 if $P(x)$ halts
0 if $P(x)$ does not halt

We designed $<Q>$ based on $<P>$ and $x$ so that:

$<Q>$ always halts $\iff <P>$ halts on input $x$

So.. if $A$ exists then we get a program $H$ for the ordinary halting problem, which we know can’t exist so $A$
can’t exist
The “Always ERROR” problem

Suppose we had a TM $E$ for the ERROR problem

1 if $Q$ always halts
0 if $Q$ does not always halt

We designed $<R>$ based on $<Q>$ so that:

$<R>$ always prints “ERROR” $\iff$ $<Q>$ always halts

So.. if $E$ exists then we get a program $A$ for the Always Halts problem, which we know can’t exist so $E$ can’t exist

Quick lessons

• Don’t rely on the idea of improved compilers and programming languages to eliminate major programming errors
  – truly safe languages can’t possibly do general computation

• Document your code!!!!
  – there is no way you can expect someone else to figure out what your program does with just your code ....since....in general it is provably impossible to do this!

A general phenomenon: Can’t tell a book by its cover

Rice’s Theorem: In general there is no way to tell anything about the input/output (I/O) behavior of a program $P$ just given its code $<P>$!
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: $p$, $q$, $r$, $s$, . . .
- 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 $p \leftrightarrow q$

English and Logic

- You cannot ride the roller coaster if you are under 4 feet tall unless you are older than 16 years old
  - $q$: you can ride the roller coaster
  - $r$: you are under 4 feet tall
  - $s$: you are older than 16

(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

\[
\begin{align*}
(p \lor \neg p)
\end{align*}
\]

\[
\begin{align*}
(p \land \neg p)
\end{align*}
\]

\[
\begin{align*}
(p \rightarrow q) \land p
\end{align*}
\]

\[
\begin{align*}
(p \land q) \lor (p \land \neg q) \lor (\neg p \land q) \lor (\neg p \land \neg q)
\end{align*}
\]
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
    - Functions
  - Several kinds of gates
  - Correspond to propositional connectives
    - Only symmetric ones (order of inputs irrelevant)

Combinational Logic Circuits

- Wires can send one value to multiple gates

A simple example: 1-bit binary adder

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

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>Cin</th>
<th>Cout</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
S = A' \ B' \ Cin + A' \ B \ Cin' + A \ B' \ Cin' + A \ B \ Cin \\
= A' \ (B' \ Cin + B \ Cin') + A \ (B' \ Cin' + B \ Cin) \\
= A' \ Z + A \ Z' \\
= A \ xor \ Z = A \ xor \ (B \ xor \ Cin)
\]
Boolean algebra

• An algebraic structure consists of
  – a set of elements B
  – binary operations \{ +, \cdot \}
  – and a unary operation \{ ' \}
  – such that the following axioms hold:

1. the set B contains at least two elements: a, b
2. closure: \( a + b \) is in B \( a \cdot b \) is in B
3. commutativity: \( a + (b + c) = (a + b) + c \)
4. identity: \( a + 0 = a \) \( a \cdot 1 = a \)
5. distributivity: \( a + (b \cdot c) = (a + b) \cdot (a + c) \)
6. complementarity: \( a + a' = 1 \) \( a \cdot a' = 0 \)

George Boole – 1854

Sum-of-products canonical forms

• Also known as disjunctive normal form
• Also known as minterm expansion

\[
\begin{array}{ccc|cc}
A & B & C & F & F' \\
\hline
0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 \\
\end{array}
\]

\[
F = A'B'C' + AB'C' + ABC' + ABC \\
F' = A'B'C' + A'BC' + AB'C + ABC'
\]

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

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

Domain: Positive Integers

Even(x) Odd(x) Prime(x) Greater(x,y) Equal(x,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
  \[ \therefore p \lor \neg p \]

• Two inference rules per binary connective one to eliminate it, one to introduce it.
  \[
  \begin{align*}
  p \land q & \quad \text{\textit{p, q}} \\
  & \quad \therefore p, q \\
  & \quad \therefore p \land q \\
  p \lor q, \neg p & \quad \text{\textit{p}} \\
  & \quad \therefore q \\
  & \quad \therefore p \lor q, q \lor p \\
  p, p \rightarrow q & \quad \text{\textit{p}} \\
  & \quad \therefore q \\
  & \quad \therefore p \rightarrow q \\
  \end{align*}
  \]

Inference Rules for Quantifiers

\[
\begin{align*}
  P(c) \text{ for some } c & \quad \therefore \exists x P(x) \\
  \forall x P(x) & \quad \therefore P(a) \text{ for any } a \\
  \end{align*}
\]

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

Even and Odd

\[
\begin{align*}
  \text{Even}(x) & \equiv \exists y (x=2y) \\
  \text{Odd}(x) & \equiv \exists y (x=2y+1) \\
  \text{Domain: Integers}
  \end{align*}
\]

• Prove: “The square of every odd number is odd”
  English proof of: \[ \forall x (\text{Odd}(x) \rightarrow \text{Odd}(x^2)) \]

Let x be an odd number.
Then \(x=2k+1\) for some integer k (depending on x)
Therefore \(x^2=(2k+1)^2=4k^2+4k+1=2(2k^2+2k)+1\).
Since \(2k^2+2k\) is an integer, \(x^2\) is odd.
Characteristic vectors

- Let $U = \{1, \ldots, 10\}$, represent the set $\{1, 3, 4, 8, 9\}$ with
  
  \[
  1011000110
  \]
- Bit operations:
  
  - $0110110100 \lor 0011010110 = 0111110110$

- `ls -l`
  
  - `drwxr-xr-x ... Documents/`
  - `rwxr-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

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

    \[
    \begin{align*}
    + & \quad 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
    0 & 1 & 2 & 3 & 4 & 5 & 6 & 0 \\
    1 & 2 & 3 & 4 & 5 & 6 & 0 & 1 \\
    2 & 3 & 4 & 5 & 6 & 0 & 1 & 2 \\
    3 & 4 & 5 & 6 & 0 & 1 & 2 & 3 \\
    4 & 5 & 6 & 0 & 1 & 2 & 3 & 4 \\
    5 & 6 & 0 & 1 & 2 & 3 & 4 & 5 \\
    6 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
    \end{align*}
    \]

    \[
    \begin{align*}
    \times & \quad 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
    0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
    1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
    2 & 0 & 2 & 4 & 6 & 1 & 3 & 5 \\
    3 & 0 & 3 & 6 & 2 & 5 & 1 & 4 \\
    4 & 0 & 4 & 1 & 5 & 2 & 6 & 3 \\
    5 & 0 & 5 & 3 & 1 & 6 & 4 & 2 \\
    6 & 0 & 6 & 5 & 4 & 3 & 2 & 1 \\
    \end{align*}
    \]

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 $a = dq + r$.

\[
q = a \div d \quad \text{and} \quad r = a \mod d
\]
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 $a + c \equiv b + d \pmod{m}$ and $ac \equiv bd \pmod{m}$.

Integer representation

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

- 99: 0110 0011, -18: 1001 0010

Two’s complement representation
Suppose $0 \leq x < 2^n$,
x is represented by the binary representation of $x$
-x is represented by the binary representation of $2^n - x$

- 99: 0110 0011, -18: 1110 1110

Hashing

- Map values from a large domain, $0 \ldots M-1$ in a much smaller domain, $0 \ldots n-1$
- Index lookup
- Test for equality
- Hash($x$) = $x \mod p$
  - (or 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

<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>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>5</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>

<table>
<thead>
<tr>
<th>$a$</th>
<th>$a^2$</th>
<th>$a^3$</th>
<th>$a^4$</th>
<th>$a^5$</th>
<th>$a^6$</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Arithmetic mod 7
**Fast exponentiation**

Repeated Squaring

---

**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 and Factoring**

\[
a = 2^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 = 46,200
\]

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

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

\[
\quad \quad \quad \quad \cdot 11^{\min(1,0)} \cdot 13^{\min(0,1)}
\]

---

**Euclid’s Algorithm**

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

```c
int GCD(int a, 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 GCD(a, m) = 1

By Bézout’s Theorem, there exist integers s and t such that sa + tm = 1.

s mod m is the multiplicative inverse of a:
1 = (sa + tm) mod m = sa mod m

Induction proofs

\[
\begin{align*}
P(0) \\
\forall k \ (P(k) \rightarrow P(k+1)) \\
\therefore \forall n \ P(n)
\end{align*}
\]

1. Prove P(0)
2. Let k be an arbitrary integer ≥ 0
3. Assume that P(k) is true
4. ...
5. Prove P(k+1) is true
6. P(k) \rightarrow P(k+1) \quad \text{Direct Proof Rule}
7. \forall k \ (P(k) \rightarrow P(k+1)) \quad \text{Intro } \forall \text{ from 2-6}
8. \forall n \ P(n) \quad \text{Induction Rule 1&7}

Strong Induction

\[
\begin{align*}
P(0) \\
\forall k ((P(0) \land P(1) \land P(2) \land \ldots \land P(k)) \rightarrow P(k+1)) \\
\therefore \forall n \ P(n)
\end{align*}
\]

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

Functions defined on rooted binary trees

- $\text{size}(\bullet) = 1$

- $\text{size}(T_1 \text{ } T_2) = 1 + \text{size}(T_1) + \text{size}(T_2)$

- $\text{height}(\bullet) = 0$

- $\text{height}(T) = 1 + \max\{\text{height}(T_1), \text{height}(T_2)\}$

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

```
statement:
{ if | "case" constant-expression | "default" | "for" | "while" } statement |
{ "if" | "else" | "switch" | "case" } statement |
{ "do" | "while" | "for" | "goto" | "continue" | "break" | "return" | "compound" | "declaration" | "expression" | "assignment-expression" | "comparision-expression" | "conditional-expression" | "logical-OR-expression" | "logical-AND-expression" | "bitwise-XOR-expression" | "bitwise-AND-expression" | "bitwise-OR-expression" | "bitwise-shift-expression" | "unary-expression" | "primary-expression" | "identifier" | "constant-expression" | "nameless-expression" } |
```

Definition of Relations

Let A and B be sets,  
A **binary relation from A to B** is a subset of A \( \times \) B  

Let A be a set,  
A **binary relation on A** is a subset of A \( \times \) A  

Let R be a relation on A  
- R is **reflexive** iff \((a,a) \in R\) for every \(a \in A\)  
- R is **symmetric** iff \((a,b) \in R\) implies \((b,a) \in R\)  
- R is **antisymmetric** iff \((a,b) \in R\) and \(a \neq b\) implies \((b,a) \in R\)  
- R is **transitive** iff \((a,b) \in R\) and \((b,c) \in R\) implies \((a,c) \in R\)  

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,  \( S \circ R \) is the relation from A to C defined  
\[ S \circ R = \{(a, c) \mid \exists b \text{ such that } (a,b) \in R \text{ and } (b,c) \in S\} \]
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

\[ R^2 = R \circ R = \{(a, c) | \exists b \text{ such that } (a, b) \in R \text{ and } (b, c) \in R\} \]

\[ R^0 = \{(a, a) | a \in A\} \]

\[ R^1 = R \]

\[ R^{n+1} = R^n \circ R \]

\[ S \circ R = \{(a, c) | \exists b \text{ such that } (a, b) \in R \text{ and } (b, c) \in S\} \]

n-ary relations

Let \( A_1, A_2, ..., A_n \) be sets. An n-ary relation on these sets is a subset of \( A_1 \times A_2 \times \ldots \times A_n \).

<table>
<thead>
<tr>
<th>Student_ID</th>
<th>Name</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>328012098</td>
<td>Knuth</td>
<td>4.00</td>
</tr>
<tr>
<td>481080220</td>
<td>Von Neuman</td>
<td>3.78</td>
</tr>
<tr>
<td>238082388</td>
<td>Russell</td>
<td>3.85</td>
</tr>
<tr>
<td>238001920</td>
<td>Einstein</td>
<td>2.11</td>
</tr>
<tr>
<td>1727015</td>
<td>Newton</td>
<td>3.61</td>
</tr>
<tr>
<td>34882811</td>
<td>Karp</td>
<td>3.98</td>
</tr>
<tr>
<td>2921938</td>
<td>Bernoulli</td>
<td>3.21</td>
</tr>
<tr>
<td>2921939</td>
<td>Bernoulli</td>
<td>3.54</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student_ID</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>328012098</td>
<td>CS</td>
</tr>
<tr>
<td>481080220</td>
<td>CS</td>
</tr>
<tr>
<td>238082388</td>
<td>Mathematics</td>
</tr>
<tr>
<td>238001920</td>
<td>Philosophy</td>
</tr>
<tr>
<td>1727017</td>
<td>Physics</td>
</tr>
<tr>
<td>34882811</td>
<td>CS</td>
</tr>
<tr>
<td>1727017</td>
<td>Physics</td>
</tr>
<tr>
<td>2921938</td>
<td>Mathematics</td>
</tr>
<tr>
<td>2921939</td>
<td>Mathematics</td>
</tr>
</tbody>
</table>

Matrix representation for relations

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

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

\[ \{(1, 1), (1, 2), (1, 4), (2, 1), (2, 3), (3, 2), (3, 3), (4, 2), (4, 3)\} \]

\[
\begin{pmatrix}
1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 \\
\end{pmatrix}
\]

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.

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>

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>State</th>
<th>L</th>
<th>R</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$s_1$</td>
<td>$s_2$</td>
<td>Beep</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_1$</td>
<td>$s_3$</td>
<td></td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_2$</td>
<td>$s_4$</td>
<td></td>
</tr>
<tr>
<td>$s_4$</td>
<td>$s_3$</td>
<td>$s_4$</td>
<td>Beep</td>
</tr>
</tbody>
</table>

Vending Machine

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

Vending Machine, Final 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

Accepts strings with a 1 three positions from the end of the string
Building a NFA from a regular expression

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

NFA to DFA: Subset construction

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 \( 0^n1^n \) is in \( B \), following \( 10^n \) after state \( p \) must lead to a final state.

But then the DFA would accept \( 0^m1^n \) 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”

| \( r_1 \) | 0. 5 1 0 0 0 0 0 0 ... |
| \( r_2 \) | 0. 3 3 3 3 3 3 3 3 ... |
| \( r_3 \) | 0. 1 4 2 5 8 5 7 1 4 ... |
| \( r_4 \) | 0. 1 4 1 5 1 9 2 6 5 ... |
| \( r_5 \) | 0. 1 2 1 2 2 5 1 2 2 ... |
| \( r_6 \) | 0. 2 5 0 0 0 6 5 0 0 ... |
| \( r_7 \) | 0. 7 1 8 2 8 1 8 5 2 ... |
| \( r_8 \) | 0. 6 1 8 0 3 3 9 4 5 ... |

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

What is a Turing Machine?

Halting Problem

• **Given:** the code of a program \( P \) and an input \( x \) for \( P \), i.e. given \( \langle P, x \rangle \)
• **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

Does $D$ halt on input $<D>$?

<table>
<thead>
<tr>
<th>Function $D(x)$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>if $H(x,x)=1$ then</td>
</tr>
<tr>
<td>while (true): /* loop forever */</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>no-op; /* do nothing and halt */</td>
</tr>
<tr>
<td>endif</td>
</tr>
</tbody>
</table>

$D$ 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 $x$]

$\iff$ $D$ 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