CSE 20—Discrete Math

Summer, 2006

July 11 (Day 2)
Number Systems/Computer Arithmetic
Predicate Logic

Instructor: Neil Rhodes

Algebraic Rules for Propositional Formulas

Equivalences between propositional formulas (similar to algebraic equivalences):

<table>
<thead>
<tr>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associative</td>
</tr>
<tr>
<td>Distributive</td>
</tr>
<tr>
<td>Double Negation</td>
</tr>
<tr>
<td>DeMorgan's</td>
</tr>
<tr>
<td>Commutative</td>
</tr>
<tr>
<td>Absorption</td>
</tr>
<tr>
<td>Bound</td>
</tr>
<tr>
<td>Idempotent</td>
</tr>
<tr>
<td>Negation</td>
</tr>
</tbody>
</table>

Boolean Functions

Functions that take \( n \) inputs (all 0 or 1) and returns a value of 0 or 1

Truth Table→Boolean Function

Disjunctive Normal Form

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### Gates

<table>
<thead>
<tr>
<th>And</th>
<th>Or</th>
<th>Not</th>
<th>Xor (exclusive or)</th>
<th>Nand</th>
<th>Nor</th>
</tr>
</thead>
</table>

### Numbers in Different Bases

Given a number \(d_n d_{n-1} d_{n-2} \ldots d_1 d_0\) in a base \(b\), then:
- Each \(d_i\) is in the range \(0 \ldots b-1\)
- The value of the number is \(d_n b^n + d_{n-1} b^{n-1} + \ldots + d_1 b + d_0\)
- \(b\) is an integer > 1
- The largest such number with \(n\) digits is \(b^n - 1\)
- The smallest such number with \(n\) digits is \(b^{n-1}\)

**Common bases in CS:**
- **2 (binary)**
  - Binary digit = bit
  - Inconvenient to read/write
- **8 (octal)**
  - No 8, 9, 10
- **16 (hexadecimal)**
  - How to represent \(d_i\) in range 10..15?
  - Very common

### Converting between Bases

**Converting a number \(n\) in base \(b\) to base \(c\):**
- Divide \(n\) by \(c\). The remainder is the rightmost digit
- Continue until \(n\) is 0

**Shortcut for converting between common bases**
- **2->8**
  - Group in triples from RHS
- **2->16**
  - Group in group of four from RHS
- **16->2**
  - Convert each hex digit into 4 binary digits
- **8->2**
  - Convert each octal digit into 3 binary digits

### Binary Numbers

**Using \(k\) bits to represent a non-negative number**
- \(k\) is often a power of 2 (8, 16, 32, 64)
- Total unique numbers is \(2^k\)
- Biggest number is \(2^k - 1\)
- Most-significant-bit
- Least-significant-bit

**Adding numbers**
- Add each pair of bits (including carry) from right to left, carrying as needed
- **Overflow**: If we end up with a carry, we’ve overflowed the result and can no longer represent it
  - \(k=8\): A8 + 86 =
How Computers Add

We want to add two binary numbers:

\[ \begin{align*}
13 + 22 & \quad \Rightarrow \quad 35
\end{align*} \]

Rules:
- \(0 + 0 = 0\)
- \(0 + 1 = 1\)
- \(1 + 0 = 1\)
- \(1 + 1 = 0\)

Half Adder (Single-bit Adder)

Truth Table:

\[
\begin{array}{c|c|c|c}
\hline
p & q & S & C \\
\hline
0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 \\
\hline
\end{array}
\]

Calculating the carry bit

\[
\begin{array}{c|c|c|c}
\hline
p & q & S & C \\
\hline
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
\hline
\end{array}
\]

Calculating the sum bit

\[
\begin{array}{c|c|c|c}
\hline
A & B & S & C \\
\hline
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 \\
\hline
\end{array}
\]

Circuit:
### Half Adder

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>S</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Overall circuit:

### Combinatorial (Combinational) Circuit

An input wire can split to be used as input to two gates
An output wire can be used as input
The outputs depend only on the present inputs
- Don’t depend on history
- One way to achieve this: no cycles
- Sequential circuits: output depends on the history

Direct correspondence between combinatorial circuits and boolean functions
- Master’s thesis by Claude Shannon: 1937

### Full Adder

Half-adder only works for rightmost bits of two numbers.

Full adder can also handle the incoming carry bit

### Negative Numbers

Sign-bit only
- MSB is now a sign bit
- Remaining bits represent the positive number
- Problems:
  - 2 representations of 0
  - Difficult to add/subtract

**Ones-complement**
- To obtain a negative number, complement (flip) each bit in the positive number
- Problems:
  - 2 representations of 0

**Twos-complement**
- To obtain the negative number, complement (flip) each bit in the positive number and add 1
- $\neg n = 2^n - n$
- $\neg n + n = 2^n - n + n = 2^n = 0$
- Negative of 0
- Problems:
  - More negative numbers than positive (not symmetric)
Adding Twos-Complement Numbers

Add the two (whether positive or negative)

If carry bit from MSB is set:
- If both were negative
  - If result looks negative, it is correct
  - Example: -3 + -2
  - If result looks positive, overflow occurred
- If both were positive
  - Not possible
- If signs were different, overflow can be ignored
  - Example: 5 + -3

If carry bit from MSB is not set:
- If both were positive and result looks negative: overflow occurred

Summary:
- Ignore the carry bit from the MSB.
- If two numbers of the same sign yield a result of a different sign, overflow occurred.

Other Operations with Twos-Complement Numbers

Subtraction: convert subtraction problem to addition problem
- Convert a - b to a + (-b)

Other Operations with Twos-Complement Numbers

Multiplication:
- Multiply by power of 2: arithmetic shift left
  - Shift (all but sign bit) 1 to the left (new bit on right is a zero)
- Arbitrary
  
  ```
  result = 0
  while b ≠ 0
      if (b and 1) ≠ 0
          result = result + a
          shift a left by one
          shift b right by one
  return result
  ```

Division
- Divide by power of 2: arithmetic shift right
  - Shift all bits 1 to the right (what about new sign bit?)
- Example of arbitrary divisor and dividend using long division: 25/4
Predicates

Grammatical origin

Alternate definitions of a predicate:
• A sentence with a finite number of variables. Replacing variables with specific values yields a statement. The domain is the set of all possible values to substitute.
• A function whose codomain is (true/false) statements.

Example:
• “\(x^2 > 1\)”
  - Domain?
  - Example statements?
• “Boy x likes girl y” (alternate syntax: )
  - Domain?
  - Example statements?

Important Sets

Z (Zahl = Number)

Z*

N (Natural)
  • No general agreement on whether it includes 0

Q (Quotient)

R (Real)

P (Prime)

Truth Set of a Predicate, P(x)

The set of all elements in the domain that make P(x) true

Denoted:
• \(\{x \in D : P(x)\}\)

Example:
• What is the truth set of “x is a factor of 8”?
  - If \(D = \{0, 1, 2\}\)?
  - If \(D = \{0, 1, 2, 3, \ldots\} (N)\)?
  - If \(D = \{-\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\} (Z)\)?

Quantifiers

Universal
• \(\forall x \in D \ S(x)\)

Existential
• \(\exists x \in D \ S(x)\)

Common to drop \(\in D\) if it is clear from context
Examples

Let $L(x, y)$ be the predicate “Boy $x$ likes girl $y$”

- $\exists y: \exists x: L(x, y)$
- $\exists y: \forall x L(x, y)$
- $\forall x \exists y: L(x, y)$
- $\forall y L(\text{"Jose"}, y)$

Negation and Quantifiers

$\neg(\exists x: P(x)) \leftrightarrow \forall x \neg P(x)$

$\neg(\forall x P(x)) \leftrightarrow \exists x: \neg P(x)$

Examples

$L(x, y) = \text{"x likes y"} \quad \text{Domain: all people in my son's 1st grade class}$

- Everyone likes Jill
- Nobody likes Jill
- Not everybody likes Jill
- There exists a person that nobody likes
- Nobody dislikes everybody
- Everybody likes him/herself
- $\forall x \forall y \forall z ( L(x, y) \land L(y, z) \rightarrow L(x, z))$
- $\forall x \forall y (L(x, y) \rightarrow \neg L(y, x))$

Converting to Predicates

If a number is a natural number, it is an integer

The square of any real number greater than 2 is greater than 4
Quantifiers and Adversaries

Statements involving quantifiers can be viewed as a game
- Two players
  - You, trying to prove the statement is true
  - An adversary, trying to prove the statement is false
- Work from left to right through the quantifiers
  - ∀y: the adversary selects an y (think of the adversary as choosing the worst possible y).
  - That y remains bound the rest of the game
  - ∃x: you select an x that will satisfy the statement. That x remains bound the rest of the game

Example:
- ∀x∈Z, ∃y∈Z: 2x = y
- ∃x∈Z: ∀y∈Z, xy = 0
- ∀y∈Z, ∃x∈Q: y≠0 → xy = 1

Limit

The limit of a sequence a is L means that a gets arbitrarily close to L

\[ \lim_{n \to \infty} a_n = L \]

∀ε ∈ R+, ∃n_0 ∈ N : ∀n ∈ N, n ≥ n_0 → L − ε ≤ a_n ≤ L + ε

Order of Growth of a Function

We say that T(n) has an asymptotic upper bound of f(n) if:

∃n_0 ∈ N : ∃c ∈ R^+ : ∀n ∈ Z, n ≥ n_0 → 0 ≤ T(n) ≤ cf(n)

Proving Quantified Statements

Prove universal statement: ∀x∈D, P(x)
- Exhaustive enumeration
- Generalizing from the generic particular
  - “Suppose x is in D”
  - …
  - Therefore P(x)
- Example: The difference of two odd numbers is even
Proving Quantified Statements

Prove existential statement: \( \exists x \in D: P(x) \)
- Constructive proof
  - Display an \( x \)
  - Give a set of directions for finding \( x \)
- Nonconstructive proof
  - Proof by contradiction (assume non-existence and show a contradiction)
  - Show \( x \) must exist
- Example:
  - In a group of 367 people, at least two share a birthday

Disproving Quantified Statements

Disprove universal statement: \( \forall x \in D, P(x) \)
- Counterexample
  - Show an \( x \) in \( D \) where not \( P(x) \)
  - Example: All primes are of the form \( 2^n - 1 \)

Disprove existential statement: \( \exists x \in D: P(x) \)
- Equivalent to:
  - Prove
  - Or, alternatively.
  - Therefore, best bet is generalizing from the generic particular.
  - Example: There exists a prime which can be written as the square of an integer > 1