“Applied Logic for HW and SW” – 2006 - present
- Boolean algebra, induction, rigorous proofs
- Reasoning about FP software and digital circuits
- Required course for CS students at OU
- Arts and Sciences students enroll occasionally
  - Literature, Linguistics, Philosophy, Physics, Chemistry, ...

Honors Program, University of Oklahoma
- Add-on for any discipline, requires 3.4 GPA (out of 4.0)
- Two “perspectives” courses required (over 20 offered)
  - AL + some FP + popular apps + term paper – Karnaugh etc
  - First technical course in “perspectives” classification
Demographics

- HCW: Logic in Action, honors perspectives course
- 36 students so far (17 in 2011, 19 in 2012)

Students by Demographics:
- Science: 44%
- Humanities: 22%
- Engineering: 31%
- Other disciplines:
  - Physics, Microbiology, Meteorology
  - Chemistry, Biochemistry, Physics, Microbiology, Meteorology
  - History, Linguistics, Economics, Psychology, Business, Philosophy, Drama, Letters
  - EE, CS, MechE, CompE, ChemE

- Highly motivated, talented

Average Exam Scores:
- 89% humanities
- 91% science
- 94% engineering

All exam scores ≥ 90% – except for 3 sci/engr and 3 humanities students
Boolean Equations

Axioms

\[
x \lor \text{False} = x
\]
\[
x \lor \text{True} = \text{True}
\]
\[
x \lor y = y \lor x
\]
\[
(x \lor y) \lor z = x \lor (y \lor z)
\]
\[
x \lor (y \land z) = (x \lor y) \land (x \lor z)
\]
\[
x \rightarrow y = (\neg x) \lor y
\]
\[
\neg(x \lor y) = (\neg x) \land (\neg y)
\]
\[
x \lor x = x
\]
\[
x \rightarrow x = \text{True}
\]
\[
\neg(\neg x) = x
\]

Derived Equations

\[
x \land \text{True} = x
\]
\[
x \land (y \lor z) = (x \land y) \lor (x \land z)
\]
\[
\neg(x \land y) = (\neg x) \lor (\neg y)
\]
\[
x \rightarrow (y \rightarrow z) = (x \land y) \rightarrow z
\]
\[
(x \land y) \lor y = y
\]
\[
\neg\neg x = x
\]
\[
\text{False} \rightarrow \text{True}
\]

... 50+ equations derived in lectures, homework, exams ...

June 2012    TFPIE    Rex Page and Ruben Gamboa    How Computers Work: Computational Thinking
Boolean Formulas $\cong$ Circuits

$$(x \lor y) \land y = y \quad \{\land \text{ absorption}\}$$

$$(x \lor y) \land y$$
$$= (x \lor y) \land (y \lor \text{False}) \quad \{\lor \text{ identity}\}$$
$$= (y \lor x) \land (y \lor \text{False}) \quad \{\lor \text{ commutative}\}$$
$$= y \lor (x \land \text{False}) \quad \{\lor \text{ distributive}\}$$
$$= y \lor \text{False} \quad \{\land \text{ null}\}$$
$$= y \quad \{\lor \text{ identity}\}$$

Derivation

Computers = Logic Engines

two domains, same methods

How Computers Work

physical representation of Boolean formula

$x$
$y$

$(\neg x) \land (\neg y)$

$x$
$y$

$\neg (x \lor y)$
Software Equations

\[
\begin{align*}
(\text{first} \ (\text{cons} \ x \ xs)) &= x \\
(\text{rest} \ (\text{cons} \ x \ xs)) &= xs \\
(\text{cons} \ x_0 \ [x_1 \ x_2 \ \ldots \ x_n]) &= [x_0 \ x_1 \ x_2 \ \ldots \ x_n] \\
(\text{len} \ \text{nil}) &= 0 \\
(\text{len} \ (\text{cons} \ x \ xs)) &= (+ \ 1 \ (\text{len} \ xs)) \\
(\text{append} \ \text{nil} \ ys) &= ys \\
(\text{append} \ (\text{cons} \ x \ xs) \ ys) &= (\text{cons} \ x \ (\text{append} \ xs \ ys)) \\
(\text{append} \ xs \ (\text{append} \ ys \ zs)) &= (\text{append} \ (\text{append} \ xs \ ys) \ zs) \\
(\text{len}(\text{append} \ xs \ ys)) &= (+ \ (\text{len} \ xs) \ (\text{len} \ ys))
\end{align*}
\]

Testable Properties

(defproperty append-is-associative)

(xs :value (random-list-of (random-integer)))
(ys :value (random-list-of (random-integer)))
(zs :value (random-list-of (random-integer)))
(equal (append xs (append ys zs))
(append (append xs ys) zs)))

Dracula Tests

(defproperty append-preserves-lens)
(xs :value (random-list-of (random-integer)))
(ys :value (random-list-of (random-integer)))
(= (\text{len}(\text{append} \ xs \ ys)) \ (+ \ (\text{len} \ xs) \ (\text{len} \ ys))))
Reasoning about Software

Axioms

\[
\begin{align*}
(first\ (cons\ x\ xs)) &= x & \{ \text{first} \} \\
(rest\ (cons\ x\ xs)) &= xs & \{ \text{rest} \} \\
(cons\ x_0\ [x_1\ x_2\ \ldots\ x_n]) &= [x_0\ x_1\ x_2\ \ldots\ x_n] & \{ \text{cons} \} \\
(len\ \Nil) &= 0 & \{ \text{len0} \} \\
(len\ (cons\ x\ xs)) &= (+\ 1\ (len\ xs)) & \{ \text{len1} \} \\
(append\ \Nil\ ys) &= ys & \{ \text{app0} \} \\
(append\ (cons\ x\ xs)\ ys) &= (cons\ x\ (append\ xs\ ys)) & \{ \text{app1} \}
\end{align*}
\]

… many other assumed properties (definitional equations) …

Derived Equations

\[
\begin{align*}
(append\ xs\ (append\ ys\ zs)) &= (append\ (append\ xs\ ys)\ zs) \\
(len\ (append\ xs\ ys)) &= (+\ (len\ xs)\ (len\ ys))
\end{align*}
\]

… many properties derived from definitional equations …

\[
\begin{align*}
(len\ (append\ [x_1\ x_2\ \ldots\ x_{n+1}]\ ys)) &= (len\ (append\ (cons\ x_1\ [x_2\ \ldots\ x_{n+1}]])\ ys)) & \{ \text{cons} \} \\
&= (len\ (cons\ x_1\ (append\ [x_2\ \ldots\ x_{n+1}]\ ys)) & \{ \text{app1} \} \\
&= (+\ 1\ (len\ (append\ [x_2\ \ldots\ x_{n+1}]\ ys))) & \{ \text{len1} \} \\
&= (+\ 1\ (+\ (len\ [x_2\ \ldots\ x_{n+1}]\ (len\ ys)))) & \{ ind\ hyp \} \\
\end{align*}
\]

… algebraic reasoning from axioms and derived equations …

\[
\begin{align*}
&= (+\ (len\ [x_1\ x_2\ \ldots\ x_{n+1}]\ (len\ ys)) & \{ cons \}
\end{align*}
\]
Mechanized Logic

Axioms

(\text{first} \ (\text{cons} \ x \ xs)) = x \quad \{\text{first}\}

(\text{rest} \ (\text{cons} \ x \ xs)) = xs \quad \{\text{rest}\}

(\text{cons} \ x_0 \ [x_1 \ x_2 \ \ldots \ x_n]) = [x_0 \ x_1 \ x_2 \ \ldots \ x_n] \quad \{\text{cons}\}

(\text{len} \ \text{nil}) = 0 \quad \{\text{len0}\}

(\text{len} \ (\text{cons} \ x \ xs)) = (+ 1 \ (\text{len} \ xs)) \quad \{\text{len1}\}

(\text{append} \ \text{nil} \ ys) = ys \quad \{\text{app0}\}

(\text{append} \ (\text{cons} \ x \ xs) \ ys) = (\text{cons} \ x \ (\text{append} \ xs \ ys)) \quad \{\text{app1}\}

ACL2 Theorems

(\text{defproperty} \ \text{append-is-associative}
(\text{x}:\text{value} \ (\text{randomlist-of} \ (\text{randominteger})))
(\text{ys}:\text{value} \ (\text{randomlist-of} \ (\text{randominteger})))
(\text{zs}:\text{value} \ (\text{randomlist-of} \ (\text{randominteger})))
(\text{equal} \ (\text{append} \ xs \ (\text{append} \ ys \ zs))
(\text{append} \ (\text{append} \ xs \ ys) \ zs)))))

(\text{defproperty} \ \text{append-preserved-length}
(\text{x}:\text{value} \ (\text{randomlist-of} \ (\text{randominteger})))
(\text{ys}:\text{value} \ (\text{randomlist-of} \ (\text{randominteger})))
(= \ (\text{len} \ (\text{append} \ xs \ ys)) \ (+ \ (\text{len} \ xs) \ (\text{len} \ ys))))

Dracula employs ACL2 theorem prover to verify properties
**Function Definitions**

```lisp
(defun bits (n) ; (bits n) = binary numeral for n
  (if (zp n) ; (numb(bits n)) = n
      nil ; {b0}
      (cons (mod n 2) (bits (floor n 2))))) ; {b1}

(defun numb (xs) ; xs = [x0, x1, ... x_w-1]
  (if (consp xs) ; (numb xs) = 2^0 x_0 + 2^1 x_1 + ... + 2^{w-1} x_{w-1}
      (numb (+ (first xs) ; {n1}
              (* 2 (numb(rest xs))))
        0)) ; {n0}
```

**Properties**

- (bits 0) = nil {b0}
- (bits (n+1)) = (cons (mod (n+1) 2) (bits (floor (n+1) 2)) {b1}
- (numb nil) = 0 {n0}
- (numb (cons x xs)) = x + 2*(numb xs) {n1}

**The 3 C’s**

- **Comprehensive** – cover all cases
- **Consistent** – no conflicts in overlapping cases
- **Computational** – circular references closer to non-circular case
Ripple-Carry Adder Circuit

Ripple-Carry Adder Circuit

input carry \( c_0 = 0 \)

2’s complement numerals for \( x \) and \( y \)
\(-2^{w-1} \leq x, y \leq 2^{w-1} - 1\)

2’s complement numeral for \( x + y \)

2’s complement arithmetic with \( w \)-bit words
Expected Property of Circuit

\[
\begin{array}{ccccccc}
\ & x_0 & y_0 & x_1 & y_1 & \cdots & x_{w-1} & y_{w-1} \\
& c_0 & & c_1 & & & c_{w-1} & \\
\downarrow & & & & & & & \\
& s_0 & & s_1 & & & s_{w-1} & \\
\end{array}
\]

\[
\begin{array}{c}
c_w \cdots c_1 c_0 \\
x_{w-1} \cdots x_1 x_0 \\
+y_{w-1} \cdots y_1 y_0 \\
c \quad s_{w-1} \cdots s_1 s_0 \\
\end{array}
\]

Expected Property:

\[
c_0 2^0 + x_{w-1} 2^{w-1} + \cdots + x_1 2^1 + x_0 2^0 + y_{w-1} 2^{w-1} + \cdots + y_1 2^1 + y_0 2^0 = c 2^w + s_{w-1} 2^{w-1} + \cdots + s_1 2^1 + s_0 2^0
\]
Algebraic Model $\equiv$ Adder Circuit

\[ \text{Expected Property} \]
\[ \begin{align*}
    \text{add}([x_0, x_1, \ldots, x_{w-1}], [y_0, y_1, \ldots, y_{w-1}], c_0) &= ([s_0, s_1, \ldots, s_{w-1}], c) \\
    \text{where} \quad (s_0, c_1) &= \text{full-adder}(x_0, y_0, c_0) \\
    ([s_1, \ldots, s_{w-1}], c) &= \text{add}([x_1, \ldots, x_{w-1}], [y_1, \ldots, y_{w-1}], c_1) \\
    \text{add}([], [], c_0) &= ([], c_0) \\
    \text{full-adder}(x, y, c_{\text{in}}) &= (s, c_{\text{out}}) \\
    \text{where} \quad s &= \text{xor-gate}(p, c_{\text{in}}) \\
    p &= \text{xor-gate}(x, y) \\
    c_{\text{out}} &= \text{or-gate}(q, r) \\
    q &= \text{and-gate}(c_{\text{in}}, p) \\
    r &= \text{and-gate}(x, y)
\end{align*} \]

\[ \text{Derive property from algebraic model} \]
Model in ACL2 Notation

\[
\text{add}(\{x_0, x_1, \ldots, x_{w-1}\}, \{y_0, y_1, \ldots, y_{w-1}\}, c_0) = (\{s_0, s_1, \ldots, s_{w-1}\}, c)
\]

where \((s_0, c_1) = \text{full-adder}(x_0, y_0, c_0)\)

\((\{s_1, \ldots, s_{w-1}\}, c) = \text{add}(\{x_1, \ldots, x_{w-1}\}, \{y_1, \ldots, y_{w-1}\}, c_1)\)

\[
\begin{align*}
\sum_{i=0}^{w-1} x_i 2^i + y_i 2^i &= (2^w + s_{w-1} 2^{w-1} + \ldots + s_1 2^1 + s_0) 2^0 \\
\end{align*}
\]

Expected Property

\[
\begin{align*}
&\text{add}, w \geq 1 \\
&\text{add}, w = 0
\end{align*}
\]

ACL2 notation

```
(defun add (x y c0) ; x = \{x_0, x_1, \ldots, x_{w-1}\}, y = \{y_0, y_1, \ldots, y_{w-1}\}
    (if (consp x) ; (add x y c0) = (\{s_0, s_1, \ldots, s_{w-1}\}, c)
        (add x y c0) = (\{s_0, s_1, \ldots, s_{w-1}\}, c)
        (let* ((x0 (first x)) (xs (rest x))
                (y0 (first y)) (ys (rest y))
                (a0 (full-adder x0 y0 c0))
                (s0 (first a0)) (c1 (second a0))
                (a (add xs ys c1))
                (ss (first a)) (c (second a))
                (list (cons s0 ss) c)) ; \{add, w \geq 1\}
            (list nil c0))) ; \{add, w = 0\}
    )
```

model in ACL2 notation
Verification by Mechanized Logic

\[
\begin{align*}
&\text{Expected Property} \\
&c_0 2^0 + x_{w-1} 2^{w-1} + \ldots + x_1 2^1 + x_0 2^0 \\
&+ y_{w-1} 2^{w-1} + \ldots + y_1 2^1 + y_0 2^0 \\
&= c_2^w + s_{w-1} 2^{w-1} + \ldots + s_1 2^1 + s_0 2^0
\end{align*}
\]

Expected Property

\[
\begin{align*}
(\text{defproperty expected-property-of-add}) \\
&\text{(c0 :value (random-between 0 1))} \\
&\text{(x :value (random-list-of (random-between 0 1))}) \\
&\text{(y :value (random-list-of (random-between 0 1) :size (len x))}) \\
&(\text{implies (= (len x) (len y))}) \\
&(\text{(let* ((a (add x y c0))}) \\
&(\text{(s (first a)) (c (second a))}) \\
&(\text{(+ (numb(list c0)) (numb x) (numb y))}) \\
&(\text{(numb(append s (list c)))))))})
\end{align*}
\]

Expected property in ACL2 notation

\[
(\text{defun numb (x) ; x = [x_0, x_1, \ldots x_{w-1}]}) \\
(\text{(if (consp x) ; (numb x) = 2^0 x_0 + 2^1 x_1 + \ldots + 2^{w-1} x_0}) \\
\text{(if (= (first x) 1) ; w \geq 1}) \\
\text{(+ 1 (* 2 (numb(rest x)))) ; x_0 = 1}) \\
\text{(* 2 (numb(rest x)))) ; x_0 = 0}) \\
\text{0)) ; w = 0})
\]
Massive Scale Computing

- Google
  - Search trees
  - Map/reduce for distributed computing
- Facebook
  - Sharding
  - Cassandra
  - NoSQL databases
- Massive scale makes engineering necessary
- Ideas described in terms of FP concepts
Course Themes and Ideas

- Central themes
  - Equations
  - Algebraic models and reasoning
- Prerequisite: basic algebra (standard college prep)

- Big ideas
  - Creativity (proofs)
  - Abstraction (functions)
  - Data (representations)
  - Algorithms (binary arith, search trees, ...)
  - Programming (3 C’s)
  - Impact (Google, Facebook)
Computational Thinking AP® Course

- Central themes
  - Equations
  - Algebraic models
- Prerequisite: basic algebra (standard college prep)
- Big ideas
  - Creativity
  - Abstraction
  - Data
  - Algorithms
  - Programming
  - Impact
  - Internet

Computational Thinking: CS Principles
- College-prep curriculum
- General education elective
- Secondary school or early college

College Board AP® Course
- Depth and substance
- Challenging, even for good students
- Accessible to all college-bound students

Positive Attributes of This Approach

June 2012  TFPIE  Rex Page and Ruben Gamboa  How Computers Work: Computational Thinking
The End