

Fourth Halmstad Summer School on Testing
 June 9-12, 2014

Testing and Verification in ACL2

Rex Page, University of Oklahoma
 June 12, 11:00 - 12:30

Half-adder circuit and formal model

+	0	1	x
0	00	01	
1	01	10	

xy	c
00	0
01	0
10	0
11	1

xy	s
00	0
01	1
10	1
11	0

half-adder circuit

```

(defun and-gate (x y)
  (if (and (= x 1) (= y 1)) 1 0))
(defun or-gate (x y)
  (if (or (= x 1) (= y 1)) 1 0))
(defun xor-gate (x y)
  (if (and (= x 1) (= y 1)) 0
      (or-gate x y)))
(defun half-adder (x y)
  (list (xor-gate x y) (and-gate x y)))
    
```

half-adder model

correctness property
 $((\text{numb}(\text{halfadder } x \ y)) = (x + y))$

Fourth Halmstad Summer School on Testing June 9-12, 2014 4

Digital circuit design verification

Commercial success for theorem provers

- ✓ AMD, Centaur Tech: ACL2
- ✓ Hewlett-Packard (in the engineering days): Isabelle
- ✓ Intel: Forte (model checking, lightweight HOL)

Digital circuits have specs

- ✓ facilitates use of formal methods
- ✓ software bug or feature?

Circuit verification

- ✓ VLSI design (eg, VHDL) - testing and fabrication
- ✓ formal model (eg, ACL2) - testing and verification
- ✓ design \equiv model ?

Small example: ripple-carry adder

- ✓ to illustrate the general idea

Fourth Halmstad Summer School on Testing - June 9-12, 2014 2

Adding binary numerals

11011101
01011101
+ 11010101
00110010

carry

Half-adder is not enough

Need 3 inputs at each stage

- ✓ 2 bits from addends
- ✓ carry bit from previous position

full adder

Fourth Halmstad Summer School on Testing June 9-12, 2014 5

Binary numerals

- ✓ Conventional rendering
 - $x_n x_{n-1} \dots x_2 x_1 x_0$
 - where each x_k is a binary digit (0 or 1)
 - x_n is high-order bit, x_0 is low-order bit
- ✓ Formal representation for our models
 - $[x_0 \ x_1 \ x_2 \ \dots \ x_n]$
 - bit-sequence in reverse order: low-order bit first, high-order last
- ✓ Converting between numerals and numbers
 - Definitional properties
 - (bits 0) = nil
 - (bits (n+1)) = (cons (mod (n+1) 2) (bits(floor (n+1) 2)))
 - (numb nil) = 0
 - (numb(cons x xs)) = x + 2*(numb xs)
 - Derivable property: numb inverts bits
 - (numb(bits n)) = n when n is a non-negative integer

```

{bits0}
{bits1}
{numb0}
{numb1}
{bits-id}
    
```

```

(defun full-adder (c-in x y)
  (let* ((h1 (half-adder x y))
         (s1 (first h1))
         (c1 (second h1))
         (h2 (half-adder s1 c-in))
         (s (first h2))
         (c2 (second h2))
         (c (or-gate c1 c2)))
    (list s c)))
    
```

correctness property
 $(\text{numb}(\text{fullAdder } c_{in} \ x \ y)) = (c_{in} + x + y)$

Fourth Halmstad Summer School on Testing June 9-12, 2014 3

Full-adder circuit

$c_{in} + x + y$	cs
0+0+0	00
0+0+1	01
0+1+0	01
0+1+1	10
1+0+0	01
1+0+1	10
1+1+0	10
1+1+1	11

$x+y$	cs
0+0	00
0+1	01
1+0	01
1+1	10

```

(defun full-adder (c-in x y)
  (let* ((h1 (half-adder x y))
         (s1 (first h1))
         (c1 (second h1))
         (h2 (half-adder s1 c-in))
         (s (first h2))
         (c2 (second h2))
         (c (or-gate c1 c2)))
    (list s c)))
    
```

correctness property
 $(\text{numb}(\text{fullAdder } c_{in} \ x \ y)) = (c_{in} + x + y)$

Fourth Halmstad Summer School on Testing June 9-12, 2014 6

w-bit ripple-carry adder
 $(\text{adder } c_0 [x_0 x_1 \dots x_{w-1}] [y_0 y_1 \dots y_{w-1}]) = [[s_0 s_1 \dots s_{w-1}] c]$

w-bit adder model (inductive)
 (defun adder (c0 x y) ; in: carry-bit and two w-bit numerals
 (if (consp x)
 (let* ((x0 (first x)) (xs (rest x)) (y0 (first y)) (ys (rest y))
 (a0 (full-adder c0 x0 y0)) (s0 (first a0)) (c1 (second a0))
 (a (adder c1 xs ys)) (ss (first a)) (c (second a)))
 (list (cons s0 ss) c)) ; {add1} ; out: w-bit numeral and carry
 (list nil c0))) ; {add0}

correctness property
 (numb(append [s₀ s₁ ... s_{w-1}] [c]))
 = (numb[x₀ x₁ ... x_{w-1}]) + (numb[y₀ y₁ ... y_{w-1}]) + c₀
 where [[s₀ s₁ ... s_{w-1}] c] = (adder c₀ [x₀ x₁ ... x_{w-1}] [y₀ y₁ ... y_{w-1}])

demo 6

7

Fourth Halmstead Summer School on Testing June 9-12, 2014

Bignum adder numerals of unbounded length

Simple problem to start: increment by 1

```

; (add-1 x) = numeral for (+ 1 (numb x)))
; (add-1 nil) = ??
{add1nil}
    
```

10

Fourth Halmstead Summer School on Testing June 9-12, 2014

Bignum adder numerals of unbounded length

8

Fourth Halmstead Summer School on Testing June 9-12, 2014

Bignum adder numerals of unbounded length

Simple problem to start: increment by 1

```

; (add-1 x) = numeral for (+ 1 (numb x)))
; (add-1 nil) = (list 1)
{add1nil}
    
```

11

Fourth Halmstead Summer School on Testing June 9-12, 2014

Bignum adder numerals of unbounded length

Simple problem to start: increment by 1

```

; (add-1 x) = numeral for (+ 1 (numb x)))
    
```

9

Fourth Halmstead Summer School on Testing June 9-12, 2014

Bignum adder numerals of unbounded length

Simple problem to start: increment by 1

```

; (add-1 x) = numeral for (+ 1 (numb x)))
; (add-1 nil) = (list 1)
; (add-1 (cons 0 x)) = ??
{add1nil}
{add10}
    
```

12

Fourth Halmstead Summer School on Testing June 9-12, 2014

Bignum adder numerals of unbounded length

Simple problem to start: increment by 1

```

; (add-1 x) = numeral for (+ 1 (numb x)))
; (add-1 nil) = (list 1)                                {add1nil}
; (add-1 (cons 0 x)) = (cons 1 x)                       {add10}

```

Fourth Halmstead Summer School on Testing June 9-12, 2014 13

Bignum adder numerals of unbounded length

Simple problem to start: increment by 1

```

; (add-1 x) = numeral for (+ 1 (numb x)))
; (add-1 nil) = (list 1)                                {add1nil}
; (add-1 (cons 0 x)) = (cons 1 x)                       {add10}
; (add-1 (cons 1 x)) = (cons 0 (add-1 x))               {add11}
(defun add-1 (x)
  (if (and (consp x) (= (first x) 1))
      (cons 0 (add-1 (rest x))) ; add11
      (cons 1 (rest x)))) ; add10

```

Fourth Halmstead Summer School on Testing June 9-12, 2014 16

Bignum adder numerals of unbounded length

Simple problem to start: increment by 1

```

; (add-1 x) = numeral for (+ 1 (numb x)))
; (add-1 nil) = (list 1)                                {add1nil}
; (add-1 (cons 0 x)) = (cons 1 x)                       {add10}
; (add-1 (cons 1 x)) = ??                               {add11}

```

Fourth Halmstead Summer School on Testing June 9-12, 2014 14

Bignum adder numerals of unbounded length

Simple problem to start: increment by 1

```

; (add-1 x) = numeral for (+ 1 (numb x)))
; (add-1 nil) = (list 1)                                {add1nil}
; (add-1 (cons 0 x)) = (cons 1 x)                       {add10}
; (add-1 (cons 1 x)) = (cons 0 (add-1 x))               {add11}
(defun add-1 (x)
  (if (and (consp x) (= (first x) 1))
      (cons 0 (add-1 (rest x))) ; add11
      (cons 1 (rest x)))) ; add10

```

Now, add a carry bit c to a numeral

```

; (add-c c x) = numeral for (+ c (numb x)))
(defun add-c (c x)
  (if (= c 1)
      (add-1 x) ; addc1
      x) ; addc0

```

Fourth Halmstead Summer School on Testing June 9-12, 2014 17

Bignum adder numerals of unbounded length

Simple problem to start: increment by 1

```

; (add-1 x) = numeral for (+ 1 (numb x)))
; (add-1 nil) = (list 1)                                {add1nil}
; (add-1 (cons 0 x)) = (cons 1 x)                       {add10}
; (add-1 (cons 1 x)) = (cons 0 (add-1 x))               {add11}

```

Fourth Halmstead Summer School on Testing June 9-12, 2014 15

Bignum adder numerals of unbounded length

Add with carry - definitional properties

```

; adder with unbounded precision
; (add c0 x y) = numeral for (+ c0 (numb x) (numb y))
; Note: (len x) may be different from (len y)

```

Fourth Halmstead Summer School on Testing June 9-12, 2014 18

Bignum adder numerals of unbounded length
Add with carry - definitional properties

```

; adder with unbounded precision
; (add c0 x y) = numeral for (+ c0 (numb x) (numb y))
; Note: (len x) may be different from (len y)
(defun add (c0 x y)
  (if (not (consp x))
      (add-c c0 y)
      (if (not (consp y))
          (add-c c0 x)
          (addx0y))))

```

properties that hold when one of the numerals is empty

... other properties (x and y non-nil) ...

Fourth Halmstead Summer School on Testing June 9-12, 2014 19

Bignum adder numerals of unbounded length
Add with carry - definitional properties

```

; adder with unbounded precision
; (add c0 x y) = numeral for (+ c0 (numb x) (numb y))
; Note: (len x) may be different from (len y)
(defun add (c0 x y)
  (if (not (consp x))
      (add-c c0 y)
      (if (not (consp y))
          (add-c c0 x)
          (let* ((x0 (first x)) ; x is not nil
                 (y0 (first y)) ; y is not nil
                 (a (full-adder c0 x0 y0))
                 (s0 (first a))
                 (c1 (second a)))
              (cons s0 (add c1 (rest x) (rest y)))))))

```

correctness property
 (numb(add c0 x y)) = c0 + (numb x) + (numb y)

demo 7

Fourth Halmstead Summer School on Testing June 9-12, 2014 22

Bignum adder numerals of unbounded length
Add with carry - definitional properties

```

; adder with unbounded precision
; (add c0 x y) = numeral for (+ c0 (numb x) (numb y))
; Note: (len x) may be different from (len y)
(defun add (c0 x y)
  (if (not (consp x))
      (add-c c0 y)
      (if (not (consp y))
          (add-c c0 x)
          (let* ((x0 (first x)) ; x is not nil
                 (y0 (first y)) ; y is not nil
                 (a (full-adder c0 x0 y0))
                 (s0 (first a))
                 (c1 (second a)))
              (cons c1 (add s0 (rest x) (rest y)))))))

```

extract low-order bits and add them

Fourth Halmstead Summer School on Testing June 9-12, 2014 20

Mechanization Is Necessary
without it, all is lost in the details

Even simple properties lead to big proofs

- ✓ millions of details in proofs of software properties
 - People can't keep track of millions of details
 - Besides, a proof at least is as likely to be wrong as a program
- ✓ people formulate properties ... computers push details
 - proof organized into lemmas — similar to software components
 - rigorous, but not fully formal
 - like a paper-and-pencil proof, as done by mathematicians
 - some lemma architectures are better than others
 - like modular decomposition of software ... design matters
 - formulation of properties is a big task
 - experience/judgment required ... as in software development

Fourth Halmstead Summer School on Testing June 9-12, 2014 23

Bignum adder numerals of unbounded length
Add with carry - definitional properties

```

; adder with unbounded precision
; (add c0 x y) = numeral for (+ c0 (numb x) (numb y))
; Note: (len x) may be different from (len y)
(defun add (c0 x y)
  (if (not (consp x))
      (add-c c0 y)
      (if (not (consp y))
          (add-c c0 x)
          (let* ((x0 (first x)) ; x is not nil
                 (y0 (first y)) ; y is not nil
                 (a (full-adder c0 x0 y0))
                 (s0 (first a))
                 (c1 (second a)))
              (cons s0 (add c1 (rest x) (rest y)))))))

```

insert low-order sum-bit into numeral for carry added to high-order bits

Fourth Halmstead Summer School on Testing June 9-12, 2014 21

When is theorem-proving practical?

Mission-critical: defect would be a catastrophe

- ✓ Intel Pentium bug in floating-point division
 - convinced AMD to spend 12 weeks with ACL2 team
 - AMD test suite for that circuit had 80-million cases
 - gazillions of potential cases ($2^{15+64} \times 2^{15+64} = 2^{158}$)
 - physically impossible to do that many tests
- ✓ NSA apparently willing to make large investments to eliminate the possibility of certain outcomes

Satisfiable properties

- ✓ Catastrophe avoided when Boolean formula holds

Resources

- ✓ VLSI design: add 10% to project budget/schedule
- ✓ software: double, triple, ... or more

Fourth Halmstead Summer School on Testing June 9-12, 2014 24

Exercises - bignum multiplier

4. Verify: `numb` inverts `bits`
5. Write defining properties for a multiplication operator for binary numerals of unbounded length
6. Define a correctness property of your multiply op
7. Use Proof Pad to run tests of the property you defined
8. Verify that the property holds for all binary numerals

hints

<u>shift and add</u> $\checkmark x_0 + 2 * (\text{numb } x) = (\text{numb}(\text{cons } x_0 \ x))$	<u>natural numbers</u> $(\text{natp } x) \equiv x \in \{0, 1, 2, \dots\}$
<u>x is even</u> $\checkmark x = 2 \lfloor x/2 \rfloor$ $\checkmark xy = 2 \lfloor x/2 \rfloor y$	<u>x is odd</u> $\checkmark x = 1 + 2 \lfloor x/2 \rfloor$ $\checkmark xy = (\text{mod } y \ 2) + 2(\lfloor y/2 \rfloor + \lfloor x/2 \rfloor y)$

Notes: http://ceres.hh.se/mediawiki/index.php/HSST_2014

Install Proof Pad: <http://proofpad.org>

Fourth Halmstead Summer School on Testing June 9-12, 2014

25

The End

June 12, 2014
11:00-12:30 session