Session 09c — 3:15-3:30, May 14

Teaching Software Correctness

May 13-15, 2008, University of Oklahoma

http://www.cs.ou.edu/~rlpage/SEcollab/tsc

Rex Page, U Oklahoma page@ou.edu

Assistants
Carl Eastlund (lead), Northeastern U cce@ccs.neu.edu
Ryan Ralston, U Oklahoma strawdog@ou.edu
Zac White, U Oklahoma zacwhite@gmail.com

National Science Foundation

Collaboration with Matthias Felleisen - NSF/DUE 0633664, 0813529, 0632872
Reasoning and Resources

Tail recursion: look ma ... no stack

or

Fibonacci six ways from Sunday

or

What I forgot to tell you ... on purpose
It's so easy, you can explain it to your grandmother

- Have conversation with your grandmother about this:
  
  ```
  qsort(x; xs) = qsort[y | y < x, y<−xs] ++ [x] ++ qsort[y | y >= x, y<−xs]
  ```

  - In 20 minutes, she'll get it
  - In 30 more minutes, she'll be able to interpret it

Now try explaining this:

```c
void quicksort(apvector<int> &array, int top, int bottom){
    if (top < bottom) {
        middle = partition(array, top, bottom);
        quicksort(array, top, middle);
        quicksort(array, middle+1, bottom);
    }
    return; }
```

How long would it take to explain this to your grandmother?

```c
int partition(apvector<int> &array, int top, int bottom){
    int x = array[top];
    int i = top - 1;
    int j = bottom + 1;
    int temp;
    do {
        do {
            j--;
        } while (x > array[j]);
        do {
            i++;
        } while (x < array[i]);
        if (i < j) {
            temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    } while (i < j);
    return j; }
```
Okay ... the Lisp version would be harder to explain

(defun before (x xs)  
  (if (endp xs)  
    nil  
    (if (< (car xs) x)  
      (cons (car xs) (before x (cdr xs)))  
      (before x (cdr xs)))))

(defun after (x xs)  
  (if (endp xs)  
    nil  
    (if (>= (car xs) x)  
      (cons (car xs) (after x (cdr xs)))  
      (after x (cdr xs)))))

(defun qsort (xs)  
  (if (endp xs)  
    nil  
    (append (qsort (before x xs))  
            (cons x (qsort (after x xs))))))

But you know that the difference is syntax. This is no more difficult for you than this

\[
\text{qsort[]} = [] \\
\text{qsort}(x: xs) = \text{qsort}[y \mid y < x, y \leftarrow xs] ++ [x] ++ \text{qsort}[y \mid y \geq x, y \leftarrow xs]
\]
Practical Value in equation-based programming

- Intel is using it to deal with many-core systems
  - They have no idea how to use 16 cores effectively
  - ... let alone 64
- Linspire uses it for Unix system admin and delivery
- Amazon.com uses it for web servers
- National Instruments uses it to verify aspects of LabVIEW
- ... and hundreds of other companies ... some small, some big
- It's a tiny percentage of the current SE picture
  - But, few software engineers have experience in things like
    - equation-based programming
    - mechanical logic
    - property-based testing
  ✓ So, there is a demand for those who do

Besides
- You'll be a better designer
- ... because you'll have a systems perspective
- You'll think big thoughts, not a quagmire of minor details
Fibonacci Numbers
an inductive definition

\[ f_0 = 0 \]
\[ f_1 = 1 \]
\[ f_{n+2} = f_{n+1} + f_n \]

\begin{verbatim}
(defun Fibonacci (n)
  (if (zp n)
      0
      (if (= n 1)
          1
          (+ (Fibonacci (- n 1))
              (Fibonacci (- n 2))))))
\end{verbatim}

transcribed to ACL2 notation
Fibonacci Numbers

inductive definition vs tail-recursive recipe

\[ f_0 = 0 \]
\[ f_1 = 1 \]
\[ f_{n+2} = f_{n+1} + f_n \]

(algebraic notation)

(defun Fibonacci (n)
  (if (zp n)
    0
    (if (= n 1)
      1
      (+ (Fibonacci (- n 1))
          (Fibonacci (- n 2))))))

(transcribed to ACL2 notation)

Tail recursive recipe

(defun fib-tail (n a b)
  (if (zp n)
      a
      (if (= n 1)
          b
          (fib-tail (- n 1) b (+ a b)))))

(defun Fibonacci-fast (n)
  (fib-tail n 0 1))

Tail-recursive call
Look ma ... no stack!
Compiled as “go to”

✓ Much better for computation
✓ More difficult to reason about
Verify: $\text{Fibonacci} = \text{Fibonacci-fast}$

*use one for reasoning, other for computation*

- $(\text{defun Fibonacci} \ (n))$
  - $(\text{if} \ (zp \ n))$
    - $0$
    - $(\text{if} \ (= \ n \ 1))$
      - $1$
      - $(+ \ (\text{Fibonacci} \ (- \ n \ 1))$
        - $(\text{Fibonacci} \ (- \ n \ 2)))$)

- $(\text{defun fib-tail} \ (n \ a \ b))$
  - $(\text{if} \ (zp \ n))$
    - $a$
    - $(\text{if} \ (= \ n \ 1))$
      - $b$
      - $(\text{fib-tail} \ (- \ n \ 1) \ b \ (+ \ a \ b))((n))$
- $(\text{defun Fibonacci-fast} \ (n))$
  - $(\text{fib-tail} \ n \ 0 \ 1))$

- $(\text{defun fib-fast-seq} \ (n \ m))$
  - $(\text{if} \ (zp \ n))$
    - $\text{nil}$
    - $(\text{cons} \ (\text{Fibonacci-fast} \ (- \ m \ n))$
      - $(\text{fib-fast-seq} \ (- \ n \ 1) \ m)))$

- $(\text{defun fib-test} \ (n))$
  - $(\text{equal} \ (\text{fib-seq} \ n \ n) \ (\text{fib-fast-seq} \ n \ n))$)

Run these to check for speed and equivalence.
Verify: Fibonacci = Fibonacci-fast
use one for reasoning, other for computation

(defun Fibonacci (n)
  (if (zp n)
    0
    (if (= n 1)
      1
      (+ (Fibonacci (- n 1))
          (Fibonacci (- n 2)))))))

(defun fib-tail (n a b)
  (if (zp n)
    a
    (if (= n 1)
      b
      (fib-tail (- n 1) b (+ a b))))))

(defun Fibonacci-fast (n)
  (fib-tail n 0 1))

(defun fib-fast-seq (n m)
  (if (zp n)
    nil
    (cons (Fibonacci-fast (- m n))
          (fib-fast-seq (- n 1) m))))

(defun fib-test (n)
  (equal (fib-seq n n) (fib-fast-seq n n)))

Functional equivalence theorem

(defthm Fibonacci=Fibonacci-fast
  (implies (natp n)
    (= (Fibonacci n) (Fibonacci-fast n))))
Verify: Fibonacci = Fibonacci-fast
use one for reasoning, other for computation

(defun Fibonacci (n)
  (if (zp n)
      0
      (if (= n 1)
          1
          (+ (Fibonacci (- n 1))
              (Fibonacci (- n 2))))))

(defun fib-seq (n m)
  (if (zp n)
      nil
      (cons (Fibonacci (- m n))
            (fib-seq (- n 1) m))))

(defun fib-tail (n a b)
  (if (zp n)
      a
      (if (= n 1)
          b
          (fib-tail (- n 1) b (+ a b)))))

(defun Fibonacci-fast (n)
  (fib-tail n 0 1))

(defun fib-fast-seq (n m)
  (if (zp n)
      nil
      (cons (Fibonacci-fast (- m n))
            (fib-fast-seq (- n 1) m))))

(defun fib-test (n)
  (equal (fib-seq n n) (fib-fast-seq n n)))

?- Functional equivalence theorem
   (include-book "arithmetic-3/extra/top-ext" :dir :system)
   (defthm Fibonacci= Fibonacci-fast
     (implies (natp n)
              (= (Fibonacci n) (Fibonacci-fast n))))
Verify: Fibonacci = Fibonacci-fast

use one for reasoning, other for computation

(defun Fibonacci (n)
  (if (zp n)
      0
      (if (= n 1)
          1
          (+ (Fibonacci (- n 1))
              (Fibonacci (- n 2))))))

(defun fib-tail (n a b)
  (if (zp n)
      a
      (if (= n 1)
          b
          (fib-tail (- n 1) b (+ a b)))))

(defun Fibonacci-fast (n)
  (fib-tail n 0 1))

☐ Functional equivalence theorem

(include-book "arithmetic-3/extra/top-ext" :dir :system)
(defthm fib-tail-satisfies-Fibonacci-recurrence
  (implies (and (natp n) (>= n 2))
    (= (fib-tail n a b)
        (+ (fib-tail (- n 1) a b) (fib-tail (- n 2) a b))))))
(defthm Fibonacci= Fibonacci-fast
  (implies (natp n)
    (= (Fibonacci n) (Fibonacci-fast n))))
Verify: Fibonacci = Fibonacci-fast

**use one for reasoning, other for computation**

(defun Fibonacci (n)
  (if (zp n) 0
    (if (= n 1) 1
      (+ (Fibonacci (- n 1)) (Fibonacci (- n 2)))))
  
(defun fib-tail (n a b)
  (if (zp n) a
    (if (= n 1) b
      (fib-tail (- n 1) b (+ a b))))
  
(defun Fibonacci-fast (n) (fib-tail n 0 1))

- **Functional equivalence theorem**
  (include-book "arithmetic-3/extra/top-ext" :dir :system)
  (defthm fib-tail-satisfies-Fibonacci-recurrence-base-case-0
    (= (fib-tail 0 a b) a))
  (defthm fib-tail-satisfies-Fibonacci-recurrence-base-case-1
    (= (fib-tail 1 a b) b))
  (defthm fib-tail-satisfies-Fibonacci-recurrence
    (implies (and (natp n) (>= n 2))
      (= (fib-tail n a b)
          (+ (fib-tail (- n 1) a b) (fib-tail (- n 2) a b)))))
  (defthm Fibonacci=Fibonacci-fast
    (implies (natp n)
      (= (Fibonacci n) (Fibonacci-fast n))))
**Fibonacci Faster**

*a cute little hack*

(defun fib-tail (n a b)
  (if (zp n)
      a
      (if (= n 1)
          b
          (fib-tail (- n 1) b (+ a b))))
  (fib-tail n 0 1))

(defun fib-tail-extra (n a b)
  (if (zp n)
      a
      (fib-tail-extra (- n 1) b (+ a b)))
  (fib-tail-extra n 0 1))

(defun Fibonacci-faster (n)
  (fib-tail-extra n 0 1))

(a cute little hack)

Half as many tests
One extra addition

How can we verify that the trick works?

(defthm Fibonacci-fast=Fibonacci-faster
  (implies (natp n)
    (= (Fibonacci-fast n) (Fibonacci-faster n))))
Fibonacci Even Faster

*a tiny bit ... a hack more for elegance than for speed*

(defun fib-tail-extra (n a b)
  (if (zp n)
      a
      (fib-tail-extra (- n 1) b (+ a b))))

(defun Fibonacci-faster (n)
  (fib-tail-extra n 0 1))

(defun fib-tail-xx (n a b)
  (if (zp n)
      b
      (fib-tail-xx (- n 1) b (+ a b))))

(defun Fibonacci-even-faster (n)
  (if (zp n)
      0
      (fib-tail-xx (- n 1) 0 1)))

Does this trick work?

(deftthm Fibonacci-faster=Fibonacci-even-faster
  (implies (natp n)
    (= (Fibonacci-faster n)
        (Fibonacci-even-faster n))))
Fibonacci Even Faster

a tiny bit ... a hack more for elegance than for speed

(defun fib-tail-extra (n a b)
  (if (zp n)
    a
    (fib-tail-extra (- n 1) b (+ a b))))

(defun Fibonacci-faster (n)
  (fib-tail-extra n 0 1))

(defun fib-tail-xx (n a b)
  (if (zp n)
    b
    (fib-tail-xx (- n 1) b (+ a b))))

(defun Fibonacci-even-faster (n)
  (if (zp n)
    0
    (fib-tail-xx (- n 1) 0 1)))

Does this trick work?

(defthm Fibonacci-faster=Fibonacci-even-faster
  (implies (natp n)
    (= (Fibonacci-faster n)
        (Fibonacci-even-faster n))))

In-class project - work in pairs
Needs lemmas:
Emulate proof of Fibonacci=Fibonacci-fast
... but go further
ACL2 source code:
http://www.cs.ou.edu/~rlpage/Secollab/tsc/Lectures/Session09Fibonacci/Fibonacci.lisp
The End