Alternative Function Definition and Memoization

ACL2 Lecture 6a

Warren A. Hunt, Jr.
hunt@cs.utexas.edu

Computer Science Department
University of Texas
2317 Speedway, M/S D9500
Austin, TX 78712-0233

May, 2021

Alternative Function Definition and Memoization

Users of ACL2 often wish to confirm that some efficient, but complex implementation, of some computing process is equivalent to a simple, clear specification.

In this talk, we exhibit several ACL2 features that demonstrate functional equivalence between functions with very different efficiencies.

We will also demonstrate ACL2's memoization facility.

Fibonacci Definition

Using ACL2, we can define the Fibonacci function.

```
(defun fib (x)
  (declare (xargs :guard (natp x)))
  (if (zp x)
      0
    (if (= x 1)
        1
      (+ (fib (- x 2))
         (fib (- x 1))))))
```

For this definition to be accepted, two termination conjectures must be checked – one for each inferior call to fib.

ACL2 can process this definition automatically, and observes that fib always returns a natural number.

Fibonacci Execution

The newly defined function, `fib`, can be executed immediately.

```
ACL2 !>(fib 10)
55
ACL2 !>(fib 20)
6765
```

However, when we evaluate the Fibonacci function with larger arguments, we must wait for the answer...

```
ACL2 !>(time$ (fib 50))
; (EV-REC *RETURN-LAST-ARG3* ...) took
; 191.77 seconds realtime, 188.34 seconds runtime
; (16 bytes allocated).
12586269025
```

Alternative Fibonacci Definition

ACL2 permits an alternative function for execution.

```
(defun fib (x)
  (declare (xargs :guard (natp x)))
  (mbe
   :logic                        :exec
   (if (zp x)                    (if (< x 10)
       0                             (case x
     (if (= x 1)                       (0 0)
         1                             (1 1)
       (+ (fib (- x 2))                (2 1)
          (fib (- x 1)))))             (3 2)
                                       (4 3)
                                       (5 5)
                                       (6 8)
                                       (7 13)
                                       (8 21)
                                       (9 34))
                                   (+ (fib (- x 2))
                                      (fib (- x 1))))))
```

Requirement for Alternative Definition

For ACL2 to accept an alternative (:exec) definition, it must prove, that the two definitions are equal when the input guard is satisfied.

```
(implies
 (natp x)
 (equal
   (if (zp x)                        (if (< x 10)
       0                                 (case x
     (if (= x 1)                           (0 0)
         1                                 (1 1)
       (+ (fib (- x 2))                    (2 1)
          (fib (- x 1)))))                 (3 2)
                                           (4 3)
                                           (5 5)
                                           (6 8)
                                           (7 13)
                                           (8 21)
                                           (9 34))
                                       (+ (fib (- x 2))
                                          (fib (- x 1))))))
```

Execution

Defined functions may be executed.

```
ACL2!>(time$ (fib 50))
; (EV-REC *RETURN-LAST-ARG3* ...) took
; 3.65 seconds realtime, 3.59 seconds runtime
; (16 bytes allocated).
12586269025
```

Wow! Now, this function call takes less than 4 seconds!

By pre-computing the first ten values – the execution speeds up considerably.

Why? Because every execution can stop when `(< X 10)` – and just return the answer from the `CASE` statement.

The `:exec` version of `FIB` includes *memoized* results for when `(< X 10)`.

Memoized Execution

We can tell ACL2 to memoize `FIB` function calls; for instance, when `(< X 40)`.

Defined functions may be executed.

```
ACL2 !>(memoize 'fib :condition '(< x 40))
  [... 65 or so lines elided ...]
```

Now, we can again evaluate our previous `(FIB 50)` call.

```
ACL2 !>(time$ (fib 50))
; (EV-REC *RETURN-LAST-ARG3* ...) took
; 0.00 seconds realtime, 0.00 seconds runtime
; (16 bytes allocated).
12586269025
```

Everything seems great! But, is it?

Memoized Execution, continued

Now, we want to calculate `(FIB 80)`, with the first 40 values memoized.

```
ACL2 !>(time$ (fib 80))
  ; (EV-REC *RETURN-LAST-ARG3* ...) took
  ; 16.90 seconds realtime, 16.82 seconds runtime
  ; (16 bytes allocated).
  23416728348467685
```

Again, we see that it takes a long time. Either, we must pre-compute all the values – and store them for future use – or we need a better approach.

What about using a new algorithm? One that recognizes the relationship between the results?

Can we idenify a recurrence relation in the sequence of results from our `FIB` function?

A New Fibonacci Function

By looking at the first ten values, we see that each entry is the sum of the preceding two entries. The first two values are given as 0 and 1.

Can we encode this relationship in a new function?

```
(defun f1 (fx-1 fx n-more)
  (declare (xargs :guard (and (natp fx-1)
                              (natp fx)
                              (natp n-more))))
  (if (zp n-more)
      fx
    (f1 fx (+ fx-1 fx) (1- n-more))))
```

Function `F1` has two *registers* and a third argument that says how many times to iterate this function.

The Completed New Fibonacci Function

We create a *wrapper* function with the first two values already computed.

```
(defun fib2 (x)
  (declare (xargs :guard (natp x)))
  (if (zp x)
      x
    (f1 0 1 (1- x))))
```

We run some tests to make sure that FIB and FIB2 agree on some values.

```
ACL2 !>(equal (fib 10) (fib2 10))
T
ACL2 !>(equal (fib 30) (fib2 30))
T
```

Looks good! So, can we compute (FIB2 100) ? Yes, in an instant!

FIB2 is Equal to FIB

So, we would like to prove this conjecture:

```
(defthm fib2-is-fib
  (implies (natp x)
           (equal (fib2 x)
                  (fib x))))
```

This observation relates the logical definitions of `FIB` and `FIB2`.

- The theorem prover uses the logical definitions to compare these functions.
- For fast evaluation, we use the `FIB2` definition.

So, now how fast is `FIB`, or more to the point, how fast is `FIB2`?

Executing FIB2

Now, we can compute `(FIB 1000)` easily by computing `(FIB2 1000)`.

```
ACL2 !>(time$ (integer-length (fib2 1000)))
; (EV-REC *RETURN-LAST-ARG3* ...) took
; 0.00 seconds realtime, 0.00 seconds runtime
; (61,200 bytes allocated).
694
```

If we want the complete answer, we can get it (which we split across 5 lines):

```
ACL2 >(FIB2 1000)
43466557686937456435688527675040625802564660517371...
78040248172908953655541794905189040387984007925516...
92959225930803226347752096896232398733224711616429...
96440906533187938298969649928516003704476137795166...
849228875
```