Pushing the While Language Challenge to Greater Depths

John Cowles
University of Wyoming
cowles@cs.uwyo.edu

Given functions:

```
base,
btm,
-----
test1,
test2,
test3,
test4,
-----
stp,
stp1,
stp2,
stp3,
stp4
```

constrained by

```
(implies (equal (base x1 x2)
                (btm))
         (equal x2 (btm)))
```

Sandip Ray: The existence of a (total) function, `G2`, satisfying the following, is consistent with the ACL2 logic.

```
;; Nested recursive depth of at most 2

(equal
  (G2 x1 x2)
  (cond ((equal x2 (btm))
            (btm))
        ((test1 x1 x2) ;; no recursion
            (base x1 x2))
        ((test2 x1 x2) ;; tail recursion
            (G2 (stp1 x1 x2)
                (stp  x1 x2)))
        (t              ;; recursive depth 2
         (let ((x3 (G2 (stp1 x1 x2)
                       (stp  x1 x2))))
            (G2 (stp2 x1 x2 x3)
                x3)))))
```

Given functions:

```
      G
      -----
      iter-stp,
      iter-stp1,
      iter-stp2

(defun              ;;iterate G n-times
  iter-G (n x1 x2)
  (if (zp n)
      x2
      (let ((x3 (G (iter-stp1 n x1 x2)
                   (iter-stp  n x1 x2))))
        (iter-G (+ -1 n)
                (iter-stp2 n x1 x2 x3)
                x3))))
```

```
(iter-G 0 x1 x2) = x2
```

```
(iter-G 1 x1 x2) = (G (iter-stp1 1 x1 x2)
                      (iter-stp  1 x1 x2))
```

```
(iter-G 2 x1 x2)
  = (G (iter-stp1
         1
         (iter-stp2
           2 x1 x2
           (G (iter-stp1 2 x1 x2)
              (iter-stp  2 x1 x2)))
         (G (iter-stp1 2 x1 x2)
            (iter-stp  2 x1 x2)))
       (iter-stp
         1
         (iter-stp2
           2 x1 x2
           (G (iter-stp1 2 x1 x2)
              (iter-stp  2 x1 x2)))
         (G (iter-stp1 2 x1 x2)
            (iter-stp  2 x1 x2))))
```

```
(mutual-recursion
 (defun
   G (x1 x2)
   (cond ((equal x2 (btm))
             (btm))
         ((test1 x1 x2) ;; no recursion
             (base x1 x2))
         ((test2 x1 x2) ;; tail recursion
             (ITER-G 1 (list x1 x2) x2))
         (t              ;; recursive depth 2
             (ITER-G 2 (list x1 x2) x2))))

 (defun                  ;;iterate G n-times
   iter-G (n x1 x2)
    (if (zp n)
        x2
        (let ((x3 (G (iter-stp1 n x1 x2)
                     (iter-stp  n x1 x2))))
          (iter-G (+ -1 n)
                  (iter-stp2 n x1 x2 x3)
                  x3))))
 ) ;; end mutual-recursion
```

Clever definitions of `iter-stp`, `iter-stp1`, `iter-stp2` in terms of `stp`, `stp1`, `stp2` leads to a proof of

```
(equal
  (G x1 x2)
  (cond ((equal x2 (btm))
         (btm))
        ((test1 x1 x2) ;; no recursion
         (base x1 x2))
        ((test2 x1 x2) ;; tail recursion
         (G (stp1 x1 x2)
            (stp  x1 x2)))
        (t             ;; recursive depth 2
         (let ((x3 (G (stp1 x1 x2)
                      (stp  x1 x2))))
         (G (stp2 x1 x2 x3)
            x3)))))
```

That is

```
(equal (G  x1 x2)
       (G2 x1 x2)
```

Achieve any desired nested recursive depth.

```
(mutual-recursion
 (defun ;; Nested recursive depth at most 4
   G (x1 x2)
   (cond ((equal x2 (btm)) * * *)
         ((test1 x1 x2) * * *);;no recursion
         ((test2 x1 x2) ;; tail recursion
            (ITER-G 1 (list x1 x2) x2))
         ((test3 x1 x2) ;; recursive depth 2
            (ITER-G 2 (list x1 x2) x2))
         ((test4 x1 x2) ;; recursive depth 3
            (ITER-G 3 (list x1 x2) x2))
         (t             ;; recursive depth 4
            (ITER-G 4 (list x1 x2) x2))))
  (defun
    iter-G (n x1 x2)    ;;iterate G n-times
    (if (zp n)
        x2
        (let ((x3 (G (iter-stp1 n x1 x2)
                     (iter-stp  n x1 x2))))
          (iter-G (+ -1 n)
                  (iter-stp2 n x1 x2 x3)
                  x3)))))
```

Rename both `G` and `iter-G` to `G$`:

```
(equal   ;; Nested recursive depth at most 4
  (G$ flg n x1 x2)
  (cond ((equal x2 (btm)) (btm))
        (flg
         (cond
          ((test1 x1 x2) * * *)
          ((test2 x1 x2) ;; tail recursion
            (G$ nil 1 (list x1 x2) x2))
          ((test3 x1 x2) ;;recursive depth 2
            (G$ nil 2 (list x1 x2) x2))
          ((test4 x1 x2) ;;recursive depth 3
            (G$ nil 3 (list x1 x2) x2))
          (t             ;;recursive depth 4
            (G$ nil 4 (list x1 x2) x2))))
        (t               ;;iterate G n-times
         (if (zp n)
             x2
             (let ((x3 (G$ t 0
                          (iter-stp1 n x1 x2)
                          (iter-stp  n x1 x2)))
               (G$ nil (+ -1 n)
                         (iter-stp2 n x1 x2 x3)
                         x3)))))))
```

KEY OBSERVATION: Equation for `G$` only has recursive depth 2.

Use FUNCTIONAL INSTANTIATION of Sandip's `G2` to show existence of a (total) function, `G$`, satisfying previous equation is consistent with the ACL2 logic.

```
(G x1 x2) = (G$ t 0 x1 x2)

(iter-G n x1 x2) = (G$ nil n x1 x2)
```

```
(defun
  G4 (x1 x2)
  (G$ t 0 x1 x2))
```

Then `G4` satisfies the next equation.

The existence of a function, `G4`, satisfying this, is consistent with the ACL2 logic.

```
;; Nested recursive depth of at most 4
(equal
  (G4 x1 x2)
  (cond ((equal x2 (btm)) (btm))
        ((test1 x1 x2) * * *);;no recursion
        ((test2 x1 x2) * * *);;tail recur.
        ((test3 x1 x2) * * *);;rec. depth 2
        ((test4 x1 x2) ;; recursive depth 3
          (let* ((x3 (G4 * * *))
                 (x4 (G4 * * *))
            (G4 (stp3 x1 x2 x3 x4)
                x4)))
        (t             ;; recursive depth 4
         (let* ((x3 (G4 (stp1 x1 x2)
                        (stp  x1 x2)))
                (x4 (G4 (stp2 x1 x2 x3)
                        x3))
                (x5 (G4 (stp3 x1 x2 x3 x4)
                        x4)))
            (G4 (stp4 x1 x2 x3 x4 x5)
                x5)))))
```

Use FUNCTIONAL INSTANTIATION of `G4` to show existence of a (total) function, `A`, satisfying this equation is consistent with the ACL2 logic.

```
;; A generalization of Ackermann's function.

(equal (A x y z)
       (if (null z)
           nil
           (cond ((equal x 0)
                  (+ y z))
                 ((equal y 0)
                  (A (+ -1 x) 1 z))
                 ((equal z 0)
                  (A x (+ -1 y) 1))
                 (t (A (+ -1 x)
                       y
                       (A x
                          (+ -1 y)
                          (A x
                             y
                             (+ -1 z))))))))
```

Use FUNCTIONAL INSTANTIATION of `G4` to show existence of a (total) function, `K91`, satisfying this equation is consistent with the ACL2 logic.

```
;; Based on Knuth's generalization
;; of McCarthy's 91 function.

(equal (K91 x)
       (cond ((null x)
              nil)
             ((> x 100)
              (+ -10 x))
             (t (K91
                  (K91
                   (K91
                    (K91 (+ 3 x))))))))
```