def::ung

By David Greve

def::ung

- defun “wrapper macro”
  - (include-book “coi/defung/defung” :dir :system)
- Admit arbitrary recursive functions
  - Without a measure
- Supports Inductive Proofs
- Efficient Execution
- Postpone Termination Proofs
- Historically Limited
  - No multi-values
  - No stobjs
  - Instability

ack

```
(def::ung ack (x y)
  (declare (xargs :signature ((integer integer) integer)))
  (if (<= x 0) (1+ y)
    (if (<= y 0) (ack (1- x) 1)
      (ack (1- x) (ack x (1- y))))))
```

rev3: “inadmissable”

```
(def::ung rev3 (x)
    (declare (xargs :signature ((true-listp) true-listp)
                    :default-value x))
    (cond
     ((endp x) nil)
     ((endp (cdr x)) (list (car x)))
     (t
      ;; a.b*.c
      (let* ((b.c       (cdr x))
             (c.rev-b   (rev3 b.c))
             (rev-b     (cdr c.rev-b))
             (b         (rev3 rev-b))
             (a         (car x))
             (a.b       (cons a b))
             (rev-b.a   (rev3 a.b))
             (c         (car c.rev-b))
             (c.rev-b.a (cons c rev-b.a)))
       c.rev-b.a))))
```

```
(defthm len-rev3
   (equal (len (rev3 x))
          (len x)))

(defthm consp-rev3
   (equal (consp (rev3 x))
          (consp x)))

(def::total rev3 (x)
   (declare (xargs :measure (len x)))
   t)
```

Recent Enhancements

- Multiple Values
- Stobjs
  - Requires :copy-args
- Observations
  - Stobjs and mv are very hard to work with
    - mv-let is not a pseudo-termp
    - pseudo-translate generates illegal bindings
    - ACL2 makes haphazard distinctions between :logic/:exec

stobj + mv

```
(defstobj st a)

(def::und copy-st (st)
  (declare (xargs :stobjs (st)
                  :signature ((stp) stp)))
  (mbe :logic (non-exec st)
       :exec (let ((a (a st)))
                   (list a))))

(def::und done (st)
  (declare (ignore st)
           (xargs :stobjs st
                  :signature ((stp) t)))
  t)

(def::und next (st)
  (declare (xargs :stobjs st
                  :signature ((stp) stp)))
  st)
```

```
(def::ung zed (n st)
  (declare (xargs :signature ((natp stp) natp stp)
                  :stobjs st
                  :copy-args (lambda (n st) (mv n (copy-st st)))))
  (if (done st) (mv n st)
    (let ((n (met ((x y) (mv (+ n 1) (+ 2 n))) (+ x y))))
      (let ((st (next st)))
        (met ((n st) (zed (1+ n) st))
          (zed (1+ n) st))))))
```

stobj copy

```
(DEFUN ZED (N ST)
  (MET ((ALT-N ALT-ST) (MV N (COPY-ST ST)))
       (MET ((DOM VAR ST) (ZED-MONADIC T N ST))
            (IF (NOT DOM)
                (NON-EXEC (MV ALT-N ALT-ST))
              (MV VAR ST)))))
```

Instability

- ACL2 is “too agressive”
  - Likes to substitute 'nil' for false variables
- Objective
  - Achieve minimal proof to admit a def::ung event
- Strategy
  - Admit a generic interpreter
  - Use meta-rules to prove final result

```
(def::ung ack-eval-fn (list term defaults defns)
  (declare (xargs :default-value (if (defung::true-fn list) nil
                                                   (enquote (cdr (assoc-equal (car term) defaults))))))
  (cond
  ;; Arguments ..
   ((defung::true-fn list)
    (if (not (consp term)) nil
      (cons (ack-eval (car term) defaults defns)
            (ack-eval-list (cdr term) defaults defns))))
  ;; Term ..
   ((atom term)   (enquote term))
   ((quotep term) (fix-quote+ term))
   ((equal (car term) 'if)
    (if (dequote (ack-eval (nth 1 term) defaults defns))
        (ack-eval (nth 2 term) defaults defns)
      (ack-eval (nth 3 term) defaults defns)))
   (t
    (let ((args (ack-eval-list (cdr term) defaults defns))
          (fn   (car term)))
      (cond
       ((consp fn)
        (ack-eval (acl2::beta-reduce-lambda-expr (cons fn args)) defaults defns))
       ((primitive-p fn)
        (base-eval fn args))
       (t
        (ack-eval (cons (cdr (assoc-equal fn defns)) args) defaults defns)))))))
```

ACL2 Wish List - 2015

- Copy state
- Internalize def::ung
- Separation of :logic/:exec
- Ability to make type-alist/linear pot explicit
- Make rewriter available to meta-functions