Preserving the Structure of Definitions After Simplification

Matt Kaufmann

UT Austin

November 6, 2018
THE PROBLEM
THE PROBLEM

**Task:** Simplify definitions (largest single task during my three-year collaboration with Kestrel; part of their APT tool suite)
THE PROBLEM

**Task:** Simplify definitions (largest single task during my three-year collaboration with Kestrel; part of their APT tool suite)

**Goal:** Preserve structure when simplifying definitions
THE PROBLEM

**Task**: Simplify definitions (largest single task during my three-year collaboration with Kestrel; part of their APT tool suite)

**Goal**: Preserve structure when simplifying definitions

**Approach**: Directed-untranslate
THE PROBLEM

Task: Simplify definitions (largest single task during my three-year collaboration with Kestrel; part of their APT tool suite)

Goal: Preserve structure when simplifying definitions

Approach: Directed-untranslate

Problem: Reconstruct \texttt{LET}, \texttt{LET*}, and \texttt{MV-LET} (and \texttt{B*}) after they are expanded away by simplification.
THE PROBLEM

Task: Simplify definitions (largest single task during my three-year collaboration with Kestrel; part of their APT tool suite)

Goal: Preserve structure when simplifying definitions

Approach: Directed-untranslate

Problem: Reconstruct LET, LET*, and MV-LET (and B*) after they are expanded away by simplification.

Solution: Make separate calls to the ACL2 rewriter while descending through the top-level IF and LAMBDA calls of the definition’s body.
(include-book "simplify")
(defun app3 (x y ign)
  (declare (ignore ign))
  (append x y))
(defun f1 (x) t)
(defun f2 (x) (f1 x))
(defun g (u)
  (let* ((temp (f2 u))
        (v temp))
    (app3 u v 17)))
EXAMPLE

(include-book "simplify")
(defun app3 (x y ign)
    (declare (ignore ign))
    (append x y))
(defstub f1 (x) t)
(defun f2 (x) (f1 x))
(defun g (u)
    (let* ((temp (f2 u))
            (v temp))
        (app3 u v 17)))

ACL2 !(simplify g)
(defun g$1 (u)
    (declare (xargs :guard t :verify-guards nil))
    (let* ((temp (f1 u)) (v temp))
        (app3 u v)))
(defthm g-becomes-g$1 (equal (g u) (g$1 u)))
(rewrite-augmented-term-rec
 aterm ; augmented term
 alist hyps geneqv thints runes ctx state)

(generalize-to-lambda formals
 rewritten-actuals
 rewritten-body)
(rewrite-augmented-term-rec
 aterm ; augmented term
 alist hyps geneqv thints runes ctx state)

(generalize-to-lambda formals
 rewritten-actuals
 rewritten-body)

(trace$
 (apt::rewrite-augmented-term-rec
 :entry (cons traced-fn (take 2 arglist))
 :exit (car (cadr values)))
 (apt::generalize-to-lambda
 :entry (cons 'generalize-to-lambda arglist)
 :exit (cons 'generalize-to-lambda values)))
Recall:

(defun g (u)
  (let* ((temp (f2 u))
          (v temp))
    (app3 u v 17)))
Recall:

(defun g (u)
  (let* ((temp (f2 u))
         (v temp))
    (app3 u v 17)))

1> (APT::REWRITE-AUGMENTED-TERM-REC
   ((LAMBDA (TEMP U)
     ((LAMBDA (V U) (APP3 U V '17))
      TEMP U))
    (F2 U) U)
   NIL)

....

<1 ((LAMBDA (TEMP U)
     ((LAMBDA (V U) (BINARY-APPEND U V))
      TEMP U))
  (F1 U) U)
ACL2 !>(untranslate
   '((LAMBDA (TEMP U)
       ((LAMBDA (V U)
           (BINARY-APPEND U V))
       TEMP U))
    (F1 U) U)
nil
(w state))
(LET* ((TEMP (F1 U)) (V TEMP))
   (APPEND U V))
ACL2 !>
1> (APT::REWRITE-AUGMENTED-TERM-REC
  ((LAMBDA (TEMP U)
    ((LAMBDA (V U) (APP3 U V '17))
     (F2 U) U)
    NIL)
  ((F2 U) U)
  NIL)
2> (APT::REWRITE-AUGMENTED-TERM-REC
  ((LAMBDA (V U) (APP3 U V '17)) TEMP U)
  ((TEMP F1 U) (U . U)))
3> (APT::REWRITE-AUGMENTED-TERM-REC
  (APP3 U V '17)
  ((V F1 U) (U . U)))
...
<3 (BINARY-APPEND U (F1 U))
<2 ((LAMBDA (V U) (BINARY-APPEND U V))
  (F1 U)
  U)
2> (APT::REWRITE-AUGMENTED-TERM-REC
   ((LAMBDA (V U) (APP3 U V '17)) TEMP U)
   ((TEMP . (F1 U)) (U . U)))
3> (APT::REWRITE-AUGMENTED-TERM-REC
   (APP3 U V '17)
   ((V F1 U) (U . U)))
<3 (BINARY-APPEND U (F1 U))
3> (GENERALIZE-TO-LAMBDA (V U)
   ((F1 U) U)
   (BINARY-APPEND U (F1 U)))
<3 (GENERALIZE-TO-LAMBDA
   ((LAMBDA (V U) (BINARY-APPEND U V))
    (F1 U)
    U)); (let ((v (f1 u))) (append u v))
<2 ((LAMBDA (V U) (BINARY-APPEND U V))
   (F1 U)
   U)
CONCLUSION

An old lesson but a good one....
CONCLUSION

An old lesson but a good one....

If an approach is problematic, try another approach!
CONCLUSION

An old lesson but a good one....

If an approach is problematic, try another approach!

DUH?
An old lesson but a good one....

If an approach is problematic, try another approach!

DUH?

If an approach (like trying to use directed-untranslate to reconstruct LET forms) is problematic, try another approach (like orchestrating calls to the rewriter that support such reconstruction)!