Adding APPLY to ACL2 – Work in Progress

Matt Kaufmann
J Strother Moore

Department of Computer Science
University of Texas at Austin

January, 2016
Motivation

Iterative constructs are common in all programming languages — except ACL2.

\[ \sum_{x \in \{1, 2, 3\}} x^2 \]

(loop for x in '(1 2 3) sum (sq x))

(sumlist '(1 2 3) 'sq)
(defun sum-sq (lst)
  (if (endp lst)
      0
      (+ (sq (car lst))
          (sum-sq (cdr lst))))))

(sum-sq '(1 2 3))
Now Write These in ACL2

\[ \sum_{x \in \{1, 2, 3\}} x^3 \]

\[ \sum_{x \in \{1, 2, 3\}} x^2 + x \]

\[ \sum_{x \in \{1, 2, 3\}} x^2 + 2x + 1 \]
Each requires a different ACL2 function, 
sum-sq, 
sum-cubes, 
sum-sq+x, 
sum-yet-another-poly.
Two Beautiful Things about Iterative Notation

Succinct: Many different computations can be described with the same control structure.

General: Lemmas can be proved about the control structure independent of the particulars.
\[
\sum_{x \in (\text{append } a \ b)} \gamma = \sum_{x \in a} \gamma + \sum_{x \in b} \gamma
\]
(sum-sq (append a b))
  = (+ (sum-sq a) (sum-sq b))

(sum-cubes (append a b))
  = (+ (sum-cubes a) (sum-cubes b))

(sum-sq+x (append a b))
  = (+ (sum-sq+x a) (sum-sq+x b))

(sum-yet-another-poly (append a b))
  = (+ (sum-yet-another-poly a)
       (sum-yet-another-poly b))
Goals

Make it possible to define such functions as:

(defun sumlist (lst fn)
  (if (endp lst)
      0
      (+ (apply fn (list (car lst)))
          (sumlist (cdr lst) fn))))
to prove and use such lemmas as:

(defthm sumlist-append
  (equal (sumlist (append a b) fn)
    (+ (sumlist a fn)
      (sumlist b fn))))
and to reason about and execute such terms as

(sumlist lst 'sq)
(sumlist lst 'cube)
(sumlist lst '(lambda (x) (+ (* x x) x)))
(sumlist lst '(lambda (x) (+ (* x x) (* 2 x) 1)))