```
#|

 Copyright (C) 1994 by Matt Kaufmann, Damir Jamsek, and Computational Logic,
 Inc.  All Rights Reserved.

 This script is hereby placed in the public domain, and therefore unlimited
 editing and redistribution is permitted.

 NO WARRANTY

 Matt Kaufmann, Damir Jamsek, and Computational Logic, Inc. PROVIDE ABSOLUTELY
 NO WARRANTY.  THE EVENT SCRIPT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
 KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED
 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE
 ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE SCRIPT IS WITH YOU.
 SHOULD THE SCRIPT PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY
 SERVICING, REPAIR OR CORRECTION.

 IN NO EVENT WILL Matt Kaufmann, Damir Jamsek, or Computational Logic, Inc. BE
 LIABLE TO YOU FOR ANY DAMAGES, ANY LOST PROFITS, LOST MONIES, OR OTHER
 SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR
 INABILITY TO USE THIS SCRIPT (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR
 DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY THIRD PARTIES), EVEN IF
 YOU HAVE ADVISED US OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY
 ANY OTHER PARTY.''

|#

;; An exercise in reverse Polish notation evaluation suggested
;; by Damir Jamsek.
```

EVENT: Start with the initial **nqthm** theory.

DEFINITION:
length (l)
$=$ **if** listp (l) **then** $1 +$ length (cdr (l))
else 0 **endif**

DEFINITION: OP1 = `'(suc)`

DEFINITION: OP2 = `'(add mul div mod)`

DEFINITION:

op1-form-p (e)
$=$ (listp (e)
$\wedge$ listp (cdr (e))
$\wedge$ (cddr (e) = **nil**)
$\wedge$ (car $(e) \in$ OP1))

DEFINITION:
op2-form-p (e)
$=$ (listp (e)
$\wedge$ listp (cdr (e))
$\wedge$ listp (cddr (e))
$\wedge$ (cdddr (e) = **nil**)
$\wedge$ (car $(e) \in$ OP2))

DEFINITION:
exp-p (e)
$=$ **if** $e \in$ **N** **then** **t**
elseif op1-form-p (e) **then** exp-p (cadr (e))
elseif op2-form-p (e) **then** exp-p (cadr (e)) $\wedge$ exp-p (caddr (e))
else **f** **endif**

DEFINITION:
eval-exp (e)
$=$ **if** $e \in$ **N** **then** e
elseif op1-form-p $(e) \wedge$ (car (e) = `'suc`)
then $1 +$ eval-exp (cadr (e))
elseif op2-form-p $(e) \wedge$ (car (e) = `'add`)
then eval-exp (cadr (e)) $+$ eval-exp (caddr (e))
elseif op2-form-p $(e) \wedge$ (car (e) = `'mul`)
then eval-exp (cadr (e)) $*$ eval-exp (caddr (e))
elseif op2-form-p $(e) \wedge$ (car (e) = `'div`)
then eval-exp (cadr (e)) $\div$ eval-exp (caddr (e))
elseif op2-form-p $(e) \wedge$ (car (e) = `'mod`)
then eval-exp (cadr (e)) **mod** eval-exp (caddr (e))
else **f** **endif**

DEFINITION:
exp-to-rpn (e)
$=$ **if** $e \in$ **N** **then** list (e)
elseif op1-form-p (e) **then** append (exp-to-rpn (cadr (e)), list (car (e)))
elseif op2-form-p (e)
then append (exp-to-rpn (caddr (e)),
append (exp-to-rpn (cadr (e)), list (car (e)))))
else **nil** **endif**

DEFINITION:

```
eval-rpn (r, s)
=   if listp (r)
    then if car (r) ∈ N then eval-rpn (cdr (r), cons (car (r), s))
         elseif car (r) = 'suc
         then eval-rpn (cdr (r), cons (1 + car (s), cdr (s)))
         elseif car (r) = 'add
         then eval-rpn (cdr (r), cons (car (s) + cadr (s), cddr (s)))
         elseif car (r) = 'mul
         then eval-rpn (cdr (r), cons (car (s) * cadr (s), cddr (s)))
         elseif car (r) = 'div
         then eval-rpn (cdr (r), cons (car (s) ÷ cadr (s), cddr (s)))
         elseif car (r) = 'mod
         then eval-rpn (cdr (r), cons (car (s) mod cadr (s), cddr (s)))
         else eval-rpn (cdr (r), s) endif
    else s endif
```

THEOREM: eval-rpn-append

eval-rpn (append (*r1*, *r2*), *s*) = eval-rpn (*r2*, eval-rpn (*r1*, *s*))

DEFINITION:

```
l1-ind (e, s)
=   if e ∈ N then t
    elseif op1-form-p (e) then l1-ind (cadr (e), s)
    elseif op2-form-p (e)
    then l1-ind (caddr (e), s)
         ∧   l1-ind (cadr (e), cons (eval-exp (caddr (e)), s))
    else t endif
```

THEOREM: l1

exp-p (*e*) → (cons (eval-exp (*e*), *s*) = eval-rpn (exp-to-rpn (*e*), *s*))

Index

eval-exp, 2, 3
eval-rpn, 3
eval-rpn-append, 3
exp-p, 2, 3
exp-to-rpn, 2, 3

l1, 3
l1-ind, 3
length, 1

op1, 1, 2
op1-form-p, 1–3
op2, 1, 2
op2-form-p, 2, 3