

#|

Copyright (C) 1994 by 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

Computational Logic, Inc. PROVIDES 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 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.

|#

:: Matt Kaufmann

:: From a session with Shaun Cooper, 12/9/91. Based on CLI Internal
:: Note 210 by Bill Young.

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

DEFINITION:

length(x)
= **if** listp(x) **then** 1 + length(cdr(x))
 else 0 **endif**

DEFINITION:

plistp(x)
= **if** listp(x) **then** plistp(cdr(x))
 else $x = \text{nil}$ **endif**

DEFINITION:

exp-p(exp)

```

=  if  $exp \in \mathbf{N}$  then t
    elseif  $\neg$  plstp( $exp$ ) then f
    elseif length( $exp$ ) = 3
    then if car( $exp$ ) = 'plus
        then  $\text{exp-p}(\text{cadr}(exp)) \wedge \text{exp-p}(\text{caddr}(exp))$ 
        elseif car( $exp$ ) = 'times
        then  $\text{exp-p}(\text{cadr}(exp)) \wedge \text{exp-p}(\text{caddr}(exp))$ 
        elseif car( $exp$ ) = 'subtract
        then  $\text{exp-p}(\text{cadr}(exp)) \wedge \text{exp-p}(\text{caddr}(exp))$ 
        else f endif
    else f endif

```

THEOREM: exp-p-opener

```

( $exp \notin \mathbf{N}$ )
→ ( $\text{exp-p}(exp)$ 
    = if  $\neg$  plstp( $exp$ ) then f
        elseif length( $exp$ ) = 3
        then if car( $exp$ ) = 'plus
            then  $\text{exp-p}(\text{cadr}(exp)) \wedge \text{exp-p}(\text{caddr}(exp))$ 
            elseif car( $exp$ ) = 'times
            then  $\text{exp-p}(\text{cadr}(exp)) \wedge \text{exp-p}(\text{caddr}(exp))$ 
            elseif car( $exp$ ) = 'subtract
            then  $\text{exp-p}(\text{cadr}(exp)) \wedge \text{exp-p}(\text{caddr}(exp))$ 
            else f endif
        else f endif)

```

DEFINITION:

```

 $\text{eval-s}(exp)$ 
= if  $\neg$   $\text{exp-p}(exp)$  then 0
    elseif  $exp \in \mathbf{N}$  then  $exp$ 
    elseif car( $exp$ ) = 'plus
    then  $\text{eval-s}(\text{cadr}(exp)) + \text{eval-s}(\text{caddr}(exp))$ 
    elseif car( $exp$ ) = 'times
    then  $\text{eval-s}(\text{cadr}(exp)) * \text{eval-s}(\text{caddr}(exp))$ 
    elseif car( $exp$ ) = 'subtract
    then  $\text{eval-s}(\text{cadr}(exp)) - \text{eval-s}(\text{caddr}(exp))$ 
    else 0 endif

```

EVENT: Disable exp-p-opener .

DEFINITION:

```

 $\text{target-inst-p}(exp)$ 
= if  $exp \simeq \text{nil}$  then  $exp \in \text{'(add mult sub)}$ 
    else plstp( $exp$ )

```

\wedge (length(*exp*) = 2)
 \wedge (car(*exp*) = 'pushc)
 \wedge (cadr(*exp*) \in \mathbf{N}) **endif**

DEFINITION:

target-inst-list-p(*exp*)
= **if** listp(*exp*)
 then target-inst-p(car(*exp*)) \wedge target-inst-list-p(cdr(*exp*))
 else *exp* = **nil** **endif**

DEFINITION:

single-step(*inst*, *s*)
= **case on** *inst*:
 case = *add*
 then cons(cadr(*s*) + car(*s*), caddr(*s*))
 case = *mult*
 then cons(cadr(*s*) * car(*s*), caddr(*s*))
 case = *sub*
 then cons(cadr(*s*) - car(*s*), caddr(*s*))
 otherwise cons(cadr(*inst*), *s*) **endcase**

DEFINITION:

interpreter-target(*inst-list*, *s*)
= **if** listp(*inst-list*)
 then interpreter-target(cdr(*inst-list*), single-step(car(*inst-list*), *s*))
 else *s* **endif**

EVENT: Enable exp-p-opener.

DEFINITION:

compile(*exp*)
= **if** \neg exp-p(*exp*) **then** **nil**
 elseif *exp* \in \mathbf{N} **then** list(list('pushc, *exp*))
 elseif car(*exp*) = 'plus
 then append(compile(cadr(*exp*)),
 append(compile(caddr(*exp*)), list('add)))
 elseif car(*exp*) = 'times
 then append(compile(cadr(*exp*)),
 append(compile(caddr(*exp*)), list('mult)))
 elseif car(*exp*) = 'subtract
 then append(compile(cadr(*exp*)),
 append(compile(caddr(*exp*)), list('sub)))
 else **nil** **endif**

EVENT: Disable exp-p-opener.

THEOREM: compile-preserves-legality
 $\text{exp-p}(exp) \rightarrow \text{target-inst-list-p}(\text{compile}(exp))$

THEOREM: interpreter-target-append
 $\text{interpreter-target}(\text{append}(inst-list1, inst-list2), s)$
 $= \text{interpreter-target}(inst-list2, \text{interpreter-target}(inst-list1, s))$

#| first version: provides too weak of an inductive hypothesis
(prove-lemma compiler-correctness (rewrite
 (implies (exp-p exp)
 (equal (eval-s exp)
 (car (interpreter-target (compile exp) s))))
|#

DEFINITION:
 $\text{compiler-correctness-induct}(exp, s)$
 $=$ **if** $\text{length}(exp) = 3$
 then $\text{compiler-correctness-induct}(\text{cadr}(exp), s)$
 \wedge $\text{compiler-correctness-induct}(\text{caddr}(exp),$
 $\text{cons}(\text{eval-s}(\text{cadr}(exp)), s))$
 else t endif

THEOREM: compiler-correctness-plus
 $\text{exp-p}(exp) \rightarrow (\text{interpreter-target}(\text{compile}(exp), s) = \text{cons}(\text{eval-s}(exp), s))$

THEOREM: compiler-correctness
 $\text{exp-p}(exp) \rightarrow (\text{eval-s}(exp) = \text{car}(\text{interpreter-target}(\text{compile}(exp), s)))$

Index

compile, 3, 4
compile-preserved-legality, 4
compiler-correctness, 4
compiler-correctness-induct, 4
compiler-correctness-plus, 4

eval-s, 2, 4
exp-p, 1–4
exp-p-opener, 2

interpreter-target, 3, 4
interpreter-target-append, 4

length, 1–4

plistp, 1, 2

single-step, 3

target-inst-list-p, 3, 4
target-inst-p, 2, 3