```
#|

 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.
|#

;; A solution to the Gilbreath card trick challenge.
;; Matt Kaufmann, 10/92.

;; The proof splits into two halves.  The lemma main-1 handles the
;; case in which we do not make the final adjustment of rotating one
;; card.  The lemma main-2 handles the other case.  We glue these
;; together in the final theorem, main.
```

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

DEFINITION:
length (x)
$=$ **if** listp (x) **then** $1 +$ length $(\mathrm{cdr}(x))$
else 0 **endif**

```
;; The following definition takes an arbitrary ‘oracle’, which says
;; whether the next card in the shuffle comes from the left pile or
;; the right pile.  See below for the definition of shuffle-top,
;; which makes the ‘move one card’ adjustment when necessary.
```

DEFINITION:
shuffle (*left*, *right*, *oracle*)
= **if** *left* $\simeq$ **nil then** *right*
elseif *right* $\simeq$ **nil then** *left*
elseif car (*oracle*)
then cons (car (*left*), shuffle (cdr (*left*), *right*, cdr (*oracle*)))
else cons (car (*right*), shuffle (*left*, cdr (*right*), cdr (*oracle*))) **endif**

```
;; To be really arbitrary, we postulate a color function that takes
;; two values (which might as well be booleans), using the
;; conservative CONSTRAIN principle to make this postulation.
```

CONSERVATIVE AXIOM: color-intro
truep (color (x)) $\vee$ falsep (color (x))

Simultaneously, we introduce the new function symbol *color*.

DEFINITION: same-color (x, y) = (color (x) $\leftrightarrow$ color (y))

DEFINITION:
altp (*pile*)
= **if** listp (*pile*)
then if listp (cdr (*pile*))
then if same-color (car (*pile*), cadr (*pile*)) **then f**
else altp (cdr (*pile*)) **endif**
else t endif
else t endif

DEFINITION:
last (x)
= **if** listp (x) $\wedge$ listp (cdr (x)) **then** last (cdr (x))
else car (x) **endif**

DEFINITION:
butlast (x)
= **if** listp (x) $\wedge$ listp (cdr (x)) **then** cons (car (x), butlast (cdr (x)))
else nil endif

DEFINITION:
shuffle-top (*left*, *right*, *oracle*)
= **let** *shuf* **be** shuffle (*left*, *right*, *oracle*)
in
if $\neg$ same-color (last (*left*), last (*right*)) **then** *shuf*
else cons (last (*shuf*), butlast (*shuf*)) **endif endlet**

DEFINITION:
even-length-p-rec (x)
$=$ **if** listp (x) **then** $\neg$ even-length-p-rec (cdr (x))
else t endif

THEOREM: even-length-p-rec-rewrite
(listp (x) $\wedge$ altp (x))
$\rightarrow$ (even-length-p-rec (x) $=$ ($\neg$ same-color (car (x), last (x))))

THEOREM: even-length-p-rec-append
even-length-p-rec (append $(x,\ y)$)
$=$ (even-length-p-rec (x) $\leftrightarrow$ even-length-p-rec (y))

```
;; A conditional rewrite rule with hypothesis
;; (and (listp x) (listp y)) would probably suffice for most of the
;; proof, but I believe that this version is needed somewhere late in
;; the proof.
```

THEOREM: altp-append
altp (append $(x,\ y)$)
$=$ (altp (x)
$\wedge$ altp (y)
$\wedge$ ((listp (x) $\wedge$ listp (y)) $\rightarrow$ ($\neg$ same-color (last (x), car (y)))))

```
;; Probably the following lemma isn't needed till the end, but let's
;; make sure that we can prove it.  Once we know this to be true,
;; we'll use it implicitly by adding in the hypothesis
;; (not (same-color (car left) (car right))) to main-1 (and similarly
;; for main-2).
```

THEOREM: last-same-color-iff-first-same-color
(listp (*left*)
$\wedge$ listp (*right*)
$\wedge$ altp (append (*left*, *right*))
$\wedge$ even-length-p-rec (append (*left*, *right*)))
$\rightarrow$ (same-color (last (*left*), last (*right*))
$\leftrightarrow$ same-color (car (*left*), car (*right*)))

```
;; Here is the induction scheme we use for main-1, and in fact also
;; for main-2 (actually main-2-lemma).
```

DEFINITION:

main-1-induction (*left*, *right*, *oracle*)
= **if** (*left* $\simeq$ **nil**) $\vee$ (*right* $\simeq$ **nil**) **then t**
elseif car (*oracle*)
then if cadr (*oracle*)
then main-1-induction (cddr (*left*), *right*, cddr (*oracle*))
else main-1-induction (cdr (*left*), cdr (*right*), cddr (*oracle*)) **endif**
elseif cadr (*oracle*)
then main-1-induction (cdr (*left*), cdr (*right*), cddr (*oracle*))
else main-1-induction (*left*, cddr (*right*), cddr (*oracle*)) **endif**

DEFINITION:
alt2-p (x)
= **if** listp (x)
then if listp (cdr (x))
then ($\neg$ same-color (car (x), cadr (x))) $\wedge$ alt2-p (cddr (x))
else f endif
else t endif

THEOREM: altp-implies-alt2-p
altp (x) $\rightarrow$ (alt2-p (x) = even-length-p-rec (x))

THEOREM: last-append
listp (y) $\rightarrow$ (last (append (x, y)) = last (y))

```
;; Now we may prove a version of the first half.
```

THEOREM: main-1
(listp (*left*)
$\wedge$ listp (*right*)
$\wedge$ even-length-p-rec (append (*left*, *right*))
$\wedge$ altp (*left*)
$\wedge$ altp (*right*)
$\wedge$ ($\neg$ same-color (car (*left*), car (*right*)))
$\wedge$ ($\neg$ same-color (last (*left*), last (*right*))))
$\rightarrow$ alt2-p (shuffle (*left*, *right*, *oracle*))

```
;; For the other half, we modify the notion of alt2-p (calling the
;; result by the weird name alt3-p), except that we expect an odd
;; number of cards.  Our strategy is to first prove that in the second
;; case, the CDR of the shuffle has this alt3-p property
;; (main-2-lemma).  Then we can show that when we move the final card
;; of the shuffle to the top, the result is an alt2-p.  The lemma
;; alt3-p-to-alt2-p below lets us do that little adjustment, once we
;; know (by the lemma shuffle-preserves-reds-equal-blacks) that the
```

```
;; shuffle has the property that the numbers of red and black cards in
;; it are the same.
```

DEFINITION:
alt3-p (x)
$=$ **if** listp (x)
then if listp (cdr (x))
then ($\neg$ same-color (car (x), cadr (x))) $\wedge$ alt3-p (cddr (x))
else t endif
else f endif

```
;; The following lemma is analogous to one proved for main-1; then the
;; proof of main-2-lemma goes through.
```

THEOREM: altp-implies-alt3-p
altp $(x) \rightarrow$ (alt3-p (x) $=$ ($\neg$ even-length-p-rec (x)))

THEOREM: main-2-lemma
(listp (*left*)
$\wedge$ listp (*right*)
$\wedge$ even-length-p-rec (append (*left*, *right*))
$\wedge$ altp (*left*)
$\wedge$ altp (*right*)
$\wedge$ same-color (car (*left*), car (*right*))
$\wedge$ same-color (last (*left*), last (*right*)))
$\rightarrow$ alt3-p (cdr (shuffle (*left*, *right*, *oracle*)))

DEFINITION:
count-color (*color*, x)
$=$ **if** listp (x)
then if *color* $=$ color (car (x)) **then** $1 +$ count-color (*color*, cdr (x))
else count-color (*color*, cdr (x)) **endif**
else 0 **endif**

DEFINITION:
reds-equal-blacks (x) $=$ (count-color (**t**, x) $=$ count-color (**f**, x))

THEOREM: alt-implies-reds-equal-blacks
altp (x)
$\rightarrow$ **if** even-length-p-rec (x) **then** count-color (**t**, x) $=$ count-color (**f**, x)
elseif color (car (x)) **then** count-color (**t**, x)
$=$ $(1 +$ count-color (**f**, x))
else $(1 +$ count-color (**t**, x)) $=$ count-color (**f**, x) **endif**

THEOREM: count-color-shuffle
$\text{count-color}\,(\mathit{color},\ \text{shuffle}\,(x,\ y,\ \mathit{oracle}))$
$= \quad (\text{count-color}\,(\mathit{color},\ x) + \text{count-color}\,(\mathit{color},\ y))$

THEOREM: shuffle-preserves-reds-equal-blacks
$(\text{altp}\,(\text{append}\,(x,\ y)) \land \text{even-length-p-rec}\,(\text{append}\,(x,\ y)))$
$\rightarrow \quad \text{reds-equal-blacks}\,(\text{shuffle}\,(x,\ y,\ \mathit{oracle}))$

THEOREM: alt3-p-to-alt2-p
$(\text{reds-equal-blacks}\,(\text{cons}\,(a,\ x)) \land \text{alt3-p}\,(x))$
$\rightarrow \quad \text{alt2-p}\,(\text{cons}\,(\text{last}\,(x),\ \text{butlast}\,(\text{cons}\,(a,\ x))))$

THEOREM: shuffle-cdr
$(\text{listp}\,(x) \land \text{listp}\,(y))$
$\rightarrow \quad (\text{listp}\,(\text{shuffle}\,(x,\ y,\ \mathit{oracle})) \land \text{listp}\,(\text{cdr}\,(\text{shuffle}\,(x,\ y,\ \mathit{oracle}))))$

THEOREM: main-2
$(\text{listp}\,(\mathit{left})$
$\land \quad \text{listp}\,(\mathit{right})$
$\land \quad \text{even-length-p-rec}\,(\text{append}\,(\mathit{left},\ \mathit{right}))$
$\land \quad \text{altp}\,(\mathit{left})$
$\land \quad \text{altp}\,(\mathit{right})$
$\land \quad \text{altp}\,(\text{append}\,(\mathit{left},\ \mathit{right}))$
$\land \quad \text{same-color}\,(\text{car}\,(\mathit{left}),\ \text{car}\,(\mathit{right}))$
$\land \quad \text{same-color}\,(\text{last}\,(\mathit{left}),\ \text{last}\,(\mathit{right})))$
$\rightarrow \quad \text{alt2-p}\,(\text{shuffle-top}\,(\mathit{left},\ \mathit{right},\ \mathit{oracle}))$

```
;; The following three events are there simply to show that our
;; definition of ``even length'' is honest.
```

DEFINITION: $\text{even-length-p}\,(x) = ((\text{length}\,(x)\ \mathbf{mod}\ 2) = 0)$

THEOREM: remainder-2-add1
$(((1 + x)\ \mathbf{mod}\ 2) = 0) = ((x\ \mathbf{mod}\ 2) \neq 0)$

THEOREM: even-length-p-is-even-length-p-rec
$\text{even-length-p}\,(x) = \text{even-length-p-rec}\,(x)$

THEOREM: main
$(\text{listp}\,(\mathit{left})$
$\land \quad \text{listp}\,(\mathit{right})$
$\land \quad \text{even-length-p}\,(\text{append}\,(\mathit{left},\ \mathit{right}))$
$\land \quad \text{altp}\,(\text{append}\,(\mathit{left},\ \mathit{right})))$
$\rightarrow \quad \text{alt2-p}\,(\text{shuffle-top}\,(\mathit{left},\ \mathit{right},\ \mathit{oracle}))$

Index

alt-implies-reds-equal-blacks, 5
alt2-p, 4, 6
alt3-p, 5, 6
alt3-p-to-alt2-p, 6
altp, 2–6
altp-append, 3
altp-implies-alt2-p, 4
altp-implies-alt3-p, 5

butlast, 2, 6

color, 2, 5
color-intro, 2
count-color, 5, 6
count-color-shuffle, 6

even-length-p, 6
even-length-p-is-even-length-p-rec, 6
even-length-p-rec, 3–6
even-length-p-rec-append, 3
even-length-p-rec-rewrite, 3

last, 2–6
last-append, 4
last-same-color-iff-first-same-color, 3
length, 1, 6

main, 6
main-1, 4
main-1-induction, 3, 4
main-2, 6
main-2-lemma, 5

reds-equal-blacks, 5, 6
remainder-2-add1, 6

same-color, 2–6
shuffle, 2, 4–6
shuffle-cdr, 6
shuffle-preserves-reds-equal-blacks, 6
shuffle-top, 2, 6