```
#|

 Copyright (C) 1994 by Computational Logic, Inc.  All Rights Reserved.
```



```
;; 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)) $\lor$ 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) $\land$ listp (cdr (x)) **then** last (cdr (x))
else car (x) **endif**

Definition:
butlast (x)
= **if** listp (x) $\land$ 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
count-color (*color*, shuffle (*x*, *y*, *oracle*))
= (count-color (*color*, *x*) + count-color (*color*, *y*))

THEOREM: shuffle-preserves-reds-equal-blacks
(altp (append (*x*, *y*)) $\wedge$ even-length-p-rec (append (*x*, *y*)))
$\rightarrow$ reds-equal-blacks (shuffle (*x*, *y*, *oracle*))

THEOREM: alt3-p-to-alt2-p
(reds-equal-blacks (cons (*a*, *x*)) $\wedge$ alt3-p (*x*))
$\rightarrow$ alt2-p (cons (last (*x*), butlast (cons (*a*, *x*))))

THEOREM: shuffle-cdr
(listp (*x*) $\wedge$ listp (*y*))
$\rightarrow$ (listp (shuffle (*x*, *y*, *oracle*)) $\wedge$ listp (cdr (shuffle (*x*, *y*, *oracle*))))

THEOREM: main-2
(listp (*left*)
$\wedge$ listp (*right*)
$\wedge$ even-length-p-rec (append (*left*, *right*))
$\wedge$ altp (*left*)
$\wedge$ altp (*right*)
$\wedge$ altp (append (*left*, *right*))
$\wedge$ same-color (car (*left*), car (*right*))
$\wedge$ same-color (last (*left*), last (*right*)))
$\rightarrow$ alt2-p (shuffle-top (*left*, *right*, *oracle*))

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

DEFINITION: even-length-p (*x*) = ((length (*x*) **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
even-length-p (*x*) = even-length-p-rec (*x*)

THEOREM: main
(listp (*left*)
$\wedge$ listp (*right*)
$\wedge$ even-length-p (append (*left*, *right*))
$\wedge$ altp (append (*left*, *right*)))
$\rightarrow$ alt2-p (shuffle-top (*left*, *right*, *oracle*))

Index