```
#|

 Copyright (C) 1994 by John Cowles.  All Rights Reserved.

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

 NO WARRANTY

 John Cowles 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 John Cowles 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.

|#
```

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

```
;                   A brief introduction to the
;                   Boyer-Moore Theorem Prover

;                               by
;                         John R. Cowles


;  The theorem prover is a computer program, written in Common Lisp and
;  about one million characters long, under continuous development since
;  1971 by B.S. Boyer and J S. Moore.  The purpose of the program is to
;  mechanize a mathematical logic suitable for the study of computation.

;  Some data types such as the nonnegative integers and the Boolean truth
;  values are built into the prover.  The user may add new recursively
;  defined data types and recursively defined functions on such data
;  types as well as prove theorems.  The prover specializes in induction
```

```
;  proofs.

;  The prover uses the prefix syntax of Lisp.  For example, the prover
;  uses (PLUS x y) where others might use PLUS( x,y ) or x + y.

;  As an example, the prover is given the task of proving the following.

;                  The SUM, from k=0 to n, of k*k!
;                                equals
;                              (n+1)! - 1.

;  First the theorem prover is initialized and arrangements are made to
;  record the proof as well as other useful information in files by the
;  command (BOOT-STRAP NQTHM) executed at the start of this file.

;  Recursively define a function that computes n!.
```

DEFINITION:
fact (n)
$=$ **if** $n \simeq 0$ **then** 1
else $n * \mathrm{fact}\,(n-1)$ **endif**

```
;  Recursively define a function, called SUM<K*FACT_K>, that computes the
;  sum on the left side of the equation given above.
```

DEFINITION:
sum<k*fact_k> (n)
$=$ **if** $n \simeq 0$ **then** 0
else sum<k*fact_k> $(n-1) + (n * \mathrm{fact}\,(n))$ **endif**

```
;  The formal argument of each of these functions is N.  The functions
;  IF, ZEROP, TIMES, SUB1, PLUS, FACT, and SUM<K*FACT_K> give the
;  following results when y and z are nonnegative integers.

;      (IF x y z)  returns  y  if x <> false
;                           z  if x =  false

;      (ZEROP y)  returns  true  if y =  0
;                          false if y <> 0

;      (TIMES y z)  returns  y * z

;      (SUB1 y)   returns  y - 1  if y > 0
```

```
;                          0      if y = 0

;      (PLUS y z)  returns  y + z

;      (FACT y)  returns  y!

;      (SUM<K*FACT_K> y)  returns  the SUM, from k=0 to y, of k*k!

;  Before the prover will accept these proposed recursive definitions for
;  the functions, FACT and SUM<K*FACT_K>, the recursion must be proved to
;  terminate.  That is, the prover verifies that functions actually exist
;  that satisfy the proposed definitions.

;  Next the prover is asked to prove the following trivial algebraic
;  modification of the theorem originally suggested above.

;                 The SUM, from k=0 to n, of k*k!
;                              plus 1
;
;                              equals
;
;                              (n+1)!.

;  The results produced by the functions EQUAL and ADD1 are given below.

;      (EQUAL x y)  returns  true  if x =  y
;                            false if x <> y

;      (ADD1 y)  returns  y + 1
```

THEOREM: sum<k*fact_k>+1=fact<n+1>

$(1 + \text{sum<k*fact\_k>}\,(n)) = \text{fact}\,(1 + n)$

```
;  After some simplification, the prover decides to use induction in the
;  proof of this lemma.

;  Now the prover is asked to prove the original version of the theorem.
;  The prover is informed that the theorem just proved is a useful hint.
```

THEOREM: sum<k*fact_k>=fact<n+1>-1

$\text{sum<k*fact\_k>}\,(n) = (\text{fact}\,(1 + n) - 1)$

```
;  With the hint, the prover has no trouble completing the proof.

;  The previous two lemmas together produce a proof by induction which
;  should be easy to follow by a person new to the theorem prover.
;  However, the hint is not needed by the prover to complete the proof of
;  the original theorem.  Let's start over and this time let the prover
;  work directly on the last lemma without first proving the first lemma.
```

EVENT: Undo back through the event named 'sum<k*fact_k>+1=fact<n+1>'.

THEOREM: sum<k*fact_k>=fact<n+1>-1
$\text{sum<k*fact\_k>}\,(n) = (\text{fact}\,(1 + n) - 1)$

```
;  This produces a mechanical proof that is much longer and no doubt more
;  mysterious to a new user of the prover.  It is also more interesting.
;  There is an induction inside an induction, some use of elimination,
;  and also some generalization.  The details of the theorem prover,
;  including induction, elimination, and generalization, are explained in:
;  R.S. Boyer and J S. Moore, A Computational Logic Handbook. Academic
;  Press, San Diego, 1988.
```

Index