```
#|

 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.
|#

; Nim game proof            Matt Wilding                   November 1991

; Requires the naturals library.
```

EVENT: Start with the library "naturals" using the compiled version.

```
#|
(Part of internal note 249)

CLI note #249:  A Verified NIM Strategy   Matt Wilding  Nov 1991

Introduction
------------

NIM is a two player game played with matches distributed into piles.
Players alternate removing at least one match from exactly one pile.
The player who removes the last match loses.

NIM is particularly interesting because it would appear to make a good
```

FM9001 stack demo program. The game has nice mathematical properties that can be verified, it's a real game that people have played for hundreds of years, and it's not I/O intensive. (In fact, no input is required to watch a game played between a "smart" player and his "random" opponent.)

A simple strategy that guarantees a win for most initial game setups has been discovered, and an NQTHM proof constructed that proves the strategy works. Subsequently, a reference to a 1901 paper by Charles Bouton that proves this same NIM strategy correct has been discovered in "Mathematical Puzzles and Diversions" by Martin Gardner.

An outline of the strategy and proof

Let n be the number of piles. Let p(i) be the number of matches in pile i. Let b(x) be the base 2 representation of x to some large number of bits. Let XOR-BV(x,y) be the bitwise exclusive or of b(x) and b(y). Let XOR-BVS(x0, x1, ... xn) be XOR-BV(x0, XOR-BV(x1, ...)).

A state is a LOSER state if

```
  XOR-BVS(p(0),...,p(n)) = b(0) and there is an i such that P(i)>1
or
  XOR-BVS(P(0),...,p(n)) = b(1) and there is no i such that P(i)>1.
```

Consider the following strategy:

If no pile has 2 matches, remove the match in one pile.

If there is exactly one pile with at least 2 matches, remove all the matches in that pile if there are an even number of non-empty piles and all but one match if there are an odd number of non-empty piles.

Otherwise, find the highest bit position n such that there is an odd number of 1 bits in the binary representation of the piles. Replace a pile with a 1 in bit position n with the "exclusive-or" of the other piles.

If not in a loser state and there are no piles with at least 2 matches, then clearly removing a non-empty pile is a valid move that will make the state a loser state.

If not in a loser state and there is exactly one pile with at least 2 matches, removing the matches in that pile if there is an even number of non-empty piles or all but one of the matches if an odd number of non-empty piles is a valid move that will make a loser state.

Otherwise, again if not in a loser state, there must be at least 2 piles with at least 2 matches and a highest bit position n such with an odd number of 1 bits. Replacing a pile with a 1 bit in bit position n in the manner described will reduce the number in that pile, so this constitutes a valid move. Also, the exclusive-or of resulting piles will be all zeros, and there will still be at least 1 pile remaining with at least 2 matches, so the resulting state will be a loser state.

Thus, the strategy transforms any non-losing state into a losing state with a valid move. Since any move from a loser state is a non-loser state, and the empty state is not a loser state, the strategy above will always yield a win if the game is in a non-losing state immediately before a turn or if the game is in a losing state immediately before the opponent's turn.

An example game

We get the theorem prover to print the evolving game state, and use r-loop to evaluate an example:

```
>(bm-trace (game-with-stupid-move :entry
    (list (if (car arglist) 'computer 'player)
          (bv-to-nat-state (cadr arglist)))))

>(r-loop)

Abbreviated Output Mode:  On
Type ? for help.
*(loser-state (nat-to-bv-state '(3 5 8) 5) 5)
 F
*(reasonable-game-statep (nat-to-bv-state '(3 5 8) 5) 5)
 T
*(game-with-stupid-move t (nat-to-bv-state '(3 5 8) 5) 5)
  1> (<<GAME-WITH-STUPID-MOVE>> '(COMPUTER (3 5 8)))
  2> (<<GAME-WITH-STUPID-MOVE>> '(PLAYER (3 5 6)))
  3> (<<GAME-WITH-STUPID-MOVE>> '(COMPUTER (2 5 6)))
  4> (<<GAME-WITH-STUPID-MOVE>> '(PLAYER (2 4 6)))
  5> (<<GAME-WITH-STUPID-MOVE>> '(COMPUTER (1 4 6)))
  6> (<<GAME-WITH-STUPID-MOVE>> '(PLAYER (1 4 5)))
```

```
  7> (<<GAME-WITH-STUPID-MOVE>> '(COMPUTER (0 4 5)))
  8> (<<GAME-WITH-STUPID-MOVE>> '(PLAYER (0 4 4)))
  9> (<<GAME-WITH-STUPID-MOVE>> '(COMPUTER (0 3 4)))
 10> (<<GAME-WITH-STUPID-MOVE>> '(PLAYER (0 3 3)))
 11> (<<GAME-WITH-STUPID-MOVE>> '(COMPUTER (0 2 3)))
 12> (<<GAME-WITH-STUPID-MOVE>> '(PLAYER (0 2 2)))
 13> (<<GAME-WITH-STUPID-MOVE>> '(COMPUTER (0 1 2)))
 14> (<<GAME-WITH-STUPID-MOVE>> '(PLAYER (0 1 0)))
 15> (<<GAME-WITH-STUPID-MOVE>> '(COMPUTER (0 0 0)))
 T
*

|#
```

DEFINITION:
put (*place*, *value*, *state*)
= **if** *place* $\simeq$ 0 **then** cons (*value*, cdr (*state*))
else cons (car (*state*), put (*place* $-$ 1, *value*, cdr (*state*))) **endif**

DEFINITION:
get (*place*, *state*)
= **if** *place* $\simeq$ 0 **then** car (*state*)
else get (*place* $-$ 1, cdr (*state*)) **endif**

THEOREM: get-put
get (*a1*, put (*a2*, *value*, *state*))
= **if** fix (*a1*) = fix (*a2*) **then** *value*
else get (*a1*, *state*) **endif**

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

EVENT: Introduce the function symbol *bv-size* of 0 arguments.

DEFINITION: bitp (*bit*) = ((*bit* = 0) $\vee$ (*bit* = 1))

DEFINITION:
bvp (*bv*)
= **if** listp (*bv*) **then** bitp (car (*bv*)) $\wedge$ bvp (cdr (*bv*))
else *bv* = **nil** **endif**

DEFINITION:
bvsp (bvs)
= **if** listp (bvs) **then** bvp (car (bvs)) $\land$ bvsp (cdr (bvs))
else bvs = **nil endif**

DEFINITION:
good-state-of-size ($state$, $size$)
= **if** listp ($state$)
then bvp (car ($state$))
$\land$ (length (car ($state$)) = fix ($size$))
$\land$ good-state-of-size (cdr ($state$), $size$)
else $state$ = **nil endif**

DEFINITION:
good-state ($state$) = good-state-of-size ($state$, BV-SIZE)

DEFINITION:
lessp-bv ($bv1$, $bv2$)
= **if** listp ($bv1$) $\land$ listp ($bv2$)
then (car ($bv1$) < car ($bv2$))
$\lor$ ((car ($bv1$) = car ($bv2$)) $\land$ lessp-bv (cdr ($bv1$), cdr ($bv2$)))
else f endif

```
;; high order bit first
```

DEFINITION:
nat-to-bv (nat, $size$)
= **if** $size \simeq 0$ **then nil**
elseif nat < exp (2, $size - 1$) **then** cons (0, nat-to-bv (nat, $size - 1$))
else cons (1, nat-to-bv (nat − exp (2, $size - 1$), $size - 1$)) **endif**

```
;; most significant bit first
```

DEFINITION:
bv-to-nat (bv)
= **if** listp (bv)
then (car (bv) $*$ exp (2, length (cdr (bv)))) + bv-to-nat (cdr (bv))
else 0 **endif**

THEOREM: length-nat-to-bv
length (nat-to-bv (nat, $size$)) = fix ($size$)

THEOREM: bv-to-nat-nat-to-bv
bv-to-nat (nat-to-bv (nat, $size$))
= **if** nat < exp (2, $size$) **then** fix (nat)
else exp (2, $size$) − 1 **endif**

THEOREM: lessp-bv-length
$\text{bvp}(x) \rightarrow (\text{bv-to-nat}(x) < \exp(2, \text{length}(x)))$

THEOREM: lessp-bv-to-nat-bv-to-nat
$((\text{length}(x) = \text{length}(y)) \land \text{bvp}(x) \land \text{bvp}(y))$
$\rightarrow \quad ((\text{bv-to-nat}(x) < \text{bv-to-nat}(y)) = \text{lessp-bv}(x, y))$

THEOREM: lessp-bv-nat-to-bv-nat-to-bv
$((x < \exp(2, \mathit{size})) \land (y < \exp(2, \mathit{size})))$
$\rightarrow \quad (\text{lessp-bv}(\text{nat-to-bv}(x, \mathit{size}), \text{nat-to-bv}(y, \mathit{size})) = (x < y))$

```
;; return the number of columns with value at least min
```

DEFINITION:
number-with-at-least (*state*, *min*, *size*)
= **if** listp (*state*)
then if ¬ lessp-bv (car (*state*), nat-to-bv (*min*, *size*))
then 1 + number-with-at-least (cdr (*state*), *min*, *size*)
else number-with-at-least (cdr (*state*), *min*, *size*) **endif**
else 0 **endif**

```
;; return a column number with value at least min
```

DEFINITION:
col-with-at-least (*state*, *min*, *size*)
= **if** listp (*state*)
then if ¬ lessp-bv (car (*state*), nat-to-bv (*min*, *size*)) **then** 0
else 1 + col-with-at-least (cdr (*state*), *min*, *size*) **endif**
else f endif

DEFINITION:
xor (*bit1*, *bit2*)
= **if** $(\mathit{bit1} \simeq 0) = (\mathit{bit2} \simeq 0)$ **then** 0
else 1 **endif**

DEFINITION:
xor-bv (*bv1*, *bv2*)
= **if** listp (*bv1*) ∧ listp (*bv2*)
then cons (xor (car (*bv1*), car (*bv2*)), xor-bv (cdr (*bv1*), cdr (*bv2*)))
else nil endif

DEFINITION:
fix-bit (x)
= **if** $x \simeq 0$ **then** 0
else 1 **endif**

DEFINITION:
fix-xor-bv (bv)
= **if** listp (bv) **then** cons (fix-bit (car (bv)), fix-xor-bv (cdr (bv)))
else nil endif

DEFINITION:
xor-bvs (bvs)
= **if** listp (bvs)
then if listp (cdr (bvs)) **then** xor-bv (car (bvs), xor-bvs (cdr (bvs)))
else fix-xor-bv (car (bvs)) **endif**
else nil endif

THEOREM: bvp-fix-xor-bv
bvp (fix-xor-bv (x))

THEOREM: bvp-fix-xor-bv-identity
bvp $(x) \rightarrow$ (fix-xor-bv $(x) = x$)

THEOREM: commutativity-of-xor
xor $(a, b) =$ xor (b, a)

THEOREM: associativity-of-xor
xor (xor (a, b), c) = xor (a, xor (b, c))

THEOREM: commutativity2-of-xor
xor (c, xor (a, b)) = xor (a, xor (c, b))

THEOREM: commutativity-of-xor-bv
xor-bv $(a, b) =$ xor-bv (b, a)

THEOREM: associativity-of-xor-bv
xor-bv (xor-bv (a, b), c) = xor-bv (a, xor-bv (b, c))

THEOREM: commutativity2-of-xor-bv
xor-bv (c, xor-bv (a, b)) = xor-bv (a, xor-bv (c, b))

```
;; return number of first vector with high bit on
```

DEFINITION:
high-bit-on (bvs)
= **if** listp (bvs)
then if caar $(bvs) =$ `1` **then** `0`
else $1 +$ high-bit-on (cdr (bvs)) **endif**
else f endif

DEFINITION:
delete-pile ($place$, $state$)
= **if** ¬ listp ($state$) **then** $state$
elseif $place \simeq 0$ **then** cdr ($state$)
else cons (car ($state$), delete-pile ($place - 1$, cdr ($state$))) **endif**

DEFINITION:
delete-high-bits ($state$)
= **if** listp ($state$) **then** cons (cdar ($state$), delete-high-bits (cdr ($state$)))
else nil endif

THEOREM: find-high-out-of-sync-rewrite
(listp ($state$) ∧ listp (car ($state$)))
→ (length (car (delete-high-bits ($state$))) < length (car ($state$)))

DEFINITION:
find-high-out-of-sync ($state$)
= **if** listp ($state$)
then if listp (car ($state$))
then if car (xor-bvs ($state$)) = 1 **then** high-bit-on ($state$)
else find-high-out-of-sync (delete-high-bits ($state$)) **endif**
else f endif
else f endif

DEFINITION:
smart-move ($state$, $size$)
= **if** number-with-at-least ($state$, 2, $size$) = 0
then cons (col-with-at-least ($state$, 1, $size$), nat-to-bv (0, $size$))
elseif number-with-at-least ($state$, 2, $size$) = 1
then cons (col-with-at-least ($state$, 2, $size$),
if (number-with-at-least ($state$, 1, $size$) **mod** 2) = 0
then nat-to-bv (0, $size$)
else nat-to-bv (1, $size$) **endif**)
else let $badcol$ **be** find-high-out-of-sync ($state$)
in
cons ($badcol$, xor-bvs (delete-pile ($badcol$, $state$))) **endlet endif**

DEFINITION:
apply-move ($move$, $state$) = put (car ($move$), cdr ($move$), $state$)

DEFINITION:
all-zeros (bv)
= **if** listp (bv) **then** (car (bv) = 0) ∧ all-zeros (cdr (bv))
else t endif

DEFINITION:
loser-state (*state*, *size*)
$=$ (((number-with-at-least (*state*, 2, *size*) $=$ 0)
  $\wedge$ ((number-with-at-least (*state*, 1, *size*) **mod** 2) $=$ 1))
 $\vee$ ((0 $<$ number-with-at-least (*state*, 2, *size*))
   $\wedge$ all-zeros (xor-bvs (*state*))))

DEFINITION:
movep (*move*, *state*, *size*)
$=$ (lessp-bv (cdr (*move*), get (car (*move*), *state*))
 $\wedge$ (length (cdr (*move*)) $=$ *size*)
 $\wedge$ bvp (cdr (*move*))
 $\wedge$ (car (*move*) $<$ length (*state*)))

```
;; the column of place has a 1 bit at the highest position
;; whose xors are not 0
```

DEFINITION:
high-bit-out-of-sync (*place*, *state*)
$=$ **if** listp (*state*)
 then if listp (car (*state*))
  then ((car (get (*place*, *state*)) $=$ 1)
    $\wedge$ (car (xor-bvs (*state*)) $=$ 1))
   $\vee$ ((car (xor-bvs (*state*)) $=$ 0)
     $\wedge$ high-bit-out-of-sync (*place*,
       delete-high-bits (*state*)))
  else f endif
 else f endif

DEFINITION:
lessp-when-high-bit-recursion (*state*, *size*)
$=$ **if** *size* $\simeq$ 0 **then t**
 else lessp-when-high-bit-recursion (delete-high-bits (*state*),
   size $-$ 1) **endif**

THEOREM: equal-length-0
(length (x) $=$ 0) $=$ ($\neg$ listp (x))

THEOREM: equal-length-1
(length (x) $=$ 1) $=$ (listp (x) $\wedge$ ($\neg$ listp (cdr (x))))

THEOREM: good-state-of-size-delete-high-bits
good-state-of-size (*state*, $1 + x$)
$\rightarrow$ good-state-of-size (delete-high-bits (*state*), x)

THEOREM: high-bit-out-of-sync-empty
$(\text{good-state-of-size}\,(\mathit{state},\ \mathit{size}) \wedge (\mathit{size} \simeq \mathtt{0}))$
$\rightarrow \quad (\neg\ \text{high-bit-out-of-sync}\,(\mathit{place},\ \mathit{state}))$

THEOREM: get-of-bad-place
$(\mathit{place} \not< \text{length}\,(\mathit{state})) \rightarrow (\text{get}\,(\mathit{place},\ \mathit{state}) = \mathtt{0})$

THEOREM: listp-delete-high-bits
$\text{listp}\,(\text{delete-high-bits}\,(x)) = \text{listp}\,(x)$

DEFINITION:
bv-not (x)
$=$ **if** $x = \mathtt{0}$ **then** $\mathtt{1}$
else $\mathtt{0}$ **endif**

THEOREM: bvp-xor-bv
$\text{bvp}\,(\text{xor-bv}\,(x,\ y))$

THEOREM: bvp-xor-bvs
$\text{bvp}\,(\text{xor-bvs}\,(\mathit{state}))$

THEOREM: equal-bitp-simplify
$(\text{bitp}\,(x) \wedge (x \neq \mathtt{0})) \rightarrow ((x = y) = (y = \mathtt{1}))$

```
;; make all bit equality constants into 0
```

THEOREM: equal-bit-1
$\text{bitp}\,(x) \rightarrow ((x = \mathtt{1}) = (x \neq \mathtt{0}))$

THEOREM: bitp-car-bvp
$\text{bvp}\,(x) \rightarrow \text{bitp}\,(\text{car}\,(x))$

THEOREM: good-state-of-size-means-bvsp
$\text{good-state-of-size}\,(\mathit{state},\ \mathit{size}) \rightarrow \text{bvsp}\,(\mathit{state})$

THEOREM: listp-xor-bv
$\text{listp}\,(\text{xor-bv}\,(x,\ y)) = (\text{listp}\,(x) \wedge \text{listp}\,(y))$

THEOREM: listp-xor-bvs
$\text{good-state-of-size}\,(\mathit{state},\ \mathit{size})$
$\rightarrow \quad (\text{listp}\,(\text{xor-bvs}\,(\mathit{state})) = (\text{listp}\,(\mathit{state}) \wedge (\mathtt{0} < \mathit{size})))$

THEOREM: bvp-get
$\text{good-state-of-size}\,(\mathit{state},\ \mathit{size})$
$\rightarrow \quad (\text{bvp}\,(\text{get}\,(\mathit{place},\ \mathit{state})) = (\mathit{place} < \text{length}\,(\mathit{state})))$

THEOREM: listp-get
$$\text{good-state-of-size}\,(\mathit{state},\ \mathit{size}) \rightarrow\ (\text{listp}\,(\text{get}\,(\mathit{place},\ \mathit{state})) = ((\mathit{place} < \text{length}\,(\mathit{state})) \land (\mathtt{0} < \mathit{size})))$$

THEOREM: car-xor-bv
$$(\text{listp}\,(x) \land \text{listp}\,(y)) \rightarrow (\text{car}\,(\text{xor-bv}\,(x,\ y)) = \text{xor}\,(\text{car}\,(x),\ \text{car}\,(y)))$$

DEFINITION:
lessp-bv-recursion (*list*, *size*)
= **if** $\mathit{size} \simeq \mathtt{0}$ **then t**
else lessp-bv-recursion (cdr (*list*), $\mathit{size} - 1$) **endif**

DEFINITION:
firstn (*list*, *n*)
= **if** $\neg$ listp (*list*) **then** *list*
elseif $n \simeq \mathtt{0}$ **then nil**
else cons (car (*list*), firstn (cdr (*list*), $n - 1$)) **endif**

DEFINITION:
min (x, y)
= **if** $x < y$ **then** fix (x)
else fix (y) **endif**

THEOREM: xor-bv-inverse
$$\text{xor-bv}\,(a,\ \text{xor-bv}\,(a,\ b)) = \text{firstn}\,(\text{fix-xor-bv}\,(b),\ \min\,(\text{length}\,(a),\ \text{length}\,(b)))$$

THEOREM: xor-bv-fix-xor-bv
$$(\text{xor-bv}\,(\text{fix-xor-bv}\,(x),\ y) = \text{xor-bv}\,(x,\ y)) \land\ (\text{xor-bv}\,(y,\ \text{fix-xor-bv}\,(x)) = \text{xor-bv}\,(y,\ x))$$

THEOREM: firstn-noop
$$(x \not< \text{length}\,(\mathit{list})) \rightarrow (\text{firstn}\,(\mathit{list},\ x) = \mathit{list})$$

THEOREM: xor-bvs-delete-high-bits
good-state-of-size (*state*, *size*)
$\rightarrow$ (xor-bvs (delete-high-bits (*state*))
= **if** ($\neg$ listp (*state*)) $\lor$ ($\mathtt{1} \not< \mathit{size}$) **then nil**
else cdr (xor-bvs (*state*)) **endif**)

THEOREM: length-xor-bvs
good-state-of-size (*state*, *size*)
$\rightarrow$ (length (xor-bvs (*state*))
= **if** listp (*state*) **then** fix (*size*)
else $\mathtt{0}$ **endif**)

THEOREM: listp-delete-pile
listp (delete-pile (*place*, *state*))
$=$ (listp (*state*) $\wedge$ ($\neg$ ((*place* $\simeq$ `0`) $\wedge$ ($\neg$ listp (cdr (*state*))))))

THEOREM: xor-bvs-delete-pile
(good-state-of-size (*state*, *size*) $\wedge$ (*place* $<$ length (*state*)))
$\rightarrow$ (xor-bvs (delete-pile (*place*, *state*))
$=$ **if** `1` $<$ length (*state*)
then xor-bv (get (*place*, *state*), xor-bvs (*state*))
else nil endif)

THEOREM: listp-delete-pile-delete-high-bits
listp (delete-pile (*place*, delete-high-bits (*state*)))
$=$ listp (delete-pile (*place*, *state*))

THEOREM: get-delete-high-bits
get (*place*, delete-high-bits (*state*)) $=$ cdr (get (*place*, *state*))

THEOREM: delete-pile-delete-high-bits
delete-pile (*place*, delete-high-bits (*state*))
$=$ delete-high-bits (delete-pile (*place*, *state*))

THEOREM: good-state-of-size-delete-pile
good-state-of-size (*state*, *size*)
$\rightarrow$ good-state-of-size (delete-pile (*place*, *state*), *size*)

THEOREM: silly-listp-cdr
(`1` $<$ length (x)) $\rightarrow$ listp (cdr (x))

THEOREM: numberp-car-bv
bvp (*bv*) $\rightarrow$ (car (*bv*) $\in \mathbf{N}$)

THEOREM: length-delete-high-bits
length (delete-high-bits (x)) $=$ length (x)

```
;; special version to help with free variable problem of next lemma
```

THEOREM: xor-bvs-delete-high-bits-2
((*size* $\neq$ `0`) $\wedge$ good-state-of-size (*state*, *size*))
$\rightarrow$ (xor-bvs (delete-high-bits (*state*))
$=$ **if** ($\neg$ listp (*state*)) $\vee$ (`1` $\not<$ *size*) **then nil**
else cdr (xor-bvs (*state*)) **endif**)

THEOREM: lessp-when-high-bit-out-of-sync-helper
(good-state-of-size (*state*, *size*)
$\wedge$ (*place* $<$ length (*state*))
$\wedge$ (`1` $<$ length (*state*))
$\wedge$ high-bit-out-of-sync (*place*, *state*))
$\rightarrow$ lessp-bv (xor-bvs (delete-pile (*place*, *state*)), get (*place*, *state*))

EVENT: Disable xor-bvs-delete-high-bits-2.

THEOREM: high-bit-out-of-sync-trivial
$(place \not< \text{length}(state)) \rightarrow (\neg\ \text{high-bit-out-of-sync}(place,\ state))$

THEOREM: lessp-when-high-bit-out-of-sync
$$\begin{aligned}
&(\text{good-state-of-size}(state,\ size) \\
&\ \wedge\ \ (1 < \text{length}(state)) \\
&\ \wedge\ \ \text{high-bit-out-of-sync}(place,\ state)) \\
&\rightarrow\ \ \text{lessp-bv}(\text{xor-bvs}(\text{delete-pile}(place,\ state)),\ \text{get}(place,\ state))
\end{aligned}$$

THEOREM: bitp-car-xor-bvs
$\text{bitp}(\text{car}(\text{xor-bvs}(bvs)))$

THEOREM: high-bit-on-works
$$\begin{aligned}
&((\text{car}(\text{xor-bvs}(state)) = 1) \wedge \text{bvsp}(state)) \\
&\rightarrow\ \ (\text{car}(\text{get}(\text{high-bit-on}(state),\ state)) = 1)
\end{aligned}$$

THEOREM: all-zeros-length-1
$(\text{length}(x) = 1) \rightarrow (\text{all-zeros}(x) = (\text{car}(x) = 0))$

THEOREM: all-zeros-xor-bvs-simple
$$\begin{aligned}
&(\text{good-state-of-size}(state,\ size) \wedge (size < 2)) \\
&\rightarrow\ \ (\text{all-zeros}(\text{xor-bvs}(state)) \\
&\qquad\ =\ \ ((size \simeq 0) \\
&\qquad\qquad \vee\ \ (\neg\ \text{listp}(state)) \\
&\qquad\qquad \vee\ \ (\text{car}(\text{xor-bvs}(state)) = 0)))
\end{aligned}$$

THEOREM: find-high-out-of-sync-works
$$\begin{aligned}
&((\neg\ \text{all-zeros}(\text{xor-bvs}(state))) \wedge \text{good-state-of-size}(state,\ size)) \\
&\rightarrow\ \ \text{high-bit-out-of-sync}(\text{find-high-out-of-sync}(state),\ state)
\end{aligned}$$

THEOREM: bvp-nat-to-bv
$\text{bvp}(\text{nat-to-bv}(nat,\ size))$

THEOREM: silly-lessp-sub1-exp-length
$((nat \not< \text{exp}(2,\ \text{length}(bv))) \wedge \text{bvp}(bv)) \rightarrow (nat \not< \text{bv-to-nat}(bv))$

DEFINITION:
$$\begin{aligned}
&\text{all-ones}(size) \\
&=\ \ \textbf{if } size \simeq 0 \ \textbf{ then nil} \\
&\qquad \textbf{else } \text{cons}(1,\ \text{all-ones}(size - 1)) \textbf{ endif}
\end{aligned}$$

THEOREM: nat-to-bv-is-all-ones
$(nat \not< \text{exp}(2,\ size)) \rightarrow (\text{nat-to-bv}(nat,\ size) = \text{all-ones}(size))$

THEOREM: length-all-ones
$\text{length}(\text{all-ones}(\mathit{size})) = \text{fix}(\mathit{size})$

THEOREM: bvp-all-ones
$\text{bvp}(\text{all-ones}(\mathit{size}))$

THEOREM: lessp-bv-all-ones-arg1
$\text{bvp}(\mathit{bv}) \rightarrow (\neg\, \text{lessp-bv}(\text{all-ones}(\mathit{size}),\ \mathit{bv}))$

THEOREM: lessp-bv-all-ones-arg2
$$\begin{array}{l} ((\text{length}(\mathit{bv}) = \mathit{size}) \wedge \text{bvp}(\mathit{bv})) \\ \rightarrow \quad (\text{lessp-bv}(\mathit{bv},\ \text{all-ones}(\mathit{size})) \\ \qquad = \quad (\text{bv-to-nat}(\mathit{bv}) < (\exp(\mathtt{2},\ \mathit{size}) - 1))) \end{array}$$

THEOREM: lessp-bv-nat-to-bv
$$\begin{array}{l} ((\text{length}(\mathit{bv}) = \text{fix}(\mathit{size})) \wedge (\mathit{nat} < \exp(\mathtt{2},\ \mathit{size})) \wedge \text{bvp}(\mathit{bv})) \\ \rightarrow \quad ((\text{lessp-bv}(\text{nat-to-bv}(\mathit{nat},\ \mathit{size}),\ \mathit{bv}) = (\mathit{nat} < \text{bv-to-nat}(\mathit{bv}))) \\ \qquad \wedge \quad (\text{lessp-bv}(\mathit{bv},\ \text{nat-to-bv}(\mathit{nat},\ \mathit{size})) = (\text{bv-to-nat}(\mathit{bv}) < \mathit{nat}))) \end{array}$$

THEOREM: length-car-state
$$\begin{array}{l} \text{good-state-of-size}(\mathit{state},\ \mathit{size}) \\ \rightarrow \quad (\text{length}(\text{car}(\mathit{state})) \\ \qquad = \quad \textbf{if } \text{listp}(\mathit{state}) \textbf{ then } \text{fix}(\mathit{size}) \\ \qquad\qquad \textbf{else } \mathtt{0} \textbf{ endif}) \end{array}$$

THEOREM: bvp-car
$(\text{bvsp}(x) \wedge \text{listp}(x)) \rightarrow \text{bvp}(\text{car}(x))$

THEOREM: lessp-bv-zero
$(\mathit{zero} \simeq \mathtt{0}) \rightarrow (\neg\, \text{lessp-bv}(x,\ \text{nat-to-bv}(\mathit{zero},\ \mathit{size})))$

THEOREM: number-with-at-least-zero
$(\mathit{zero} \simeq \mathtt{0}) \rightarrow (\text{number-with-at-least}(\mathit{bv},\ \mathit{zero},\ \mathit{size}) = \text{length}(\mathit{bv}))$

THEOREM: lessp-1-exp
$(\mathtt{1} < \exp(x,\ y)) = ((\mathtt{1} < x) \wedge (y \not\simeq \mathtt{0}))$

THEOREM: lessp-x-exp-x-y
$(x < \exp(x,\ y)) = (((x \simeq \mathtt{0}) \wedge (y \simeq \mathtt{0})) \vee ((\mathtt{1} < x) \wedge (\mathtt{1} < y)))$

THEOREM: lessp-bv-col-get-with-at-least
$$\begin{array}{l} ((x < y) \\ \ \wedge \quad (y < \exp(\mathtt{2},\ \mathit{size})) \\ \ \wedge \quad \text{good-state-of-size}(\mathit{state},\ \mathit{size}) \\ \ \wedge \quad (\text{number-with-at-least}(\mathit{state},\ y,\ \mathit{size}) \not\simeq \mathtt{0})) \\ \rightarrow \quad \text{lessp-bv}(\text{nat-to-bv}(x,\ \mathit{size}),\ \text{get}(\text{col-with-at-least}(\mathit{state},\ y,\ \mathit{size}),\ \mathit{state})) \end{array}$$

THEOREM: lessp-1-x-means-not-zerop-x
$(1 < x) \rightarrow (x \not\simeq 0)$

THEOREM: length-xor-bvs-delete-pile
$((1 < \text{length}\,(\mathit{state})) \land \text{good-state-of-size}\,(\mathit{state},\ \mathit{size}))$
$\rightarrow \quad (\text{length}\,(\text{xor-bvs}\,(\text{delete-pile}\,(n,\ \mathit{state}))) = \text{fix}\,(\mathit{size}))$

THEOREM: high-bit-on-reasonable
$((\text{car}\,(\text{xor-bvs}\,(\mathit{state})) = 1) \land \text{good-state-of-size}\,(\mathit{state},\ \mathit{size}))$
$\rightarrow \quad (\text{high-bit-on}\,(\mathit{state}) < \text{length}\,(\mathit{state}))$

THEOREM: find-high-out-of-sync-reasonable
$(\text{listp}\,(\mathit{state}) \land \text{good-state-of-size}\,(\mathit{state},\ \mathit{size}))$
$\rightarrow \quad (\text{find-high-out-of-sync}\,(\mathit{state}) < \text{length}\,(\mathit{state}))$

THEOREM: col-with-at-least-reasonable
$((\text{number-with-at-least}\,(\mathit{state},\ n,\ \mathit{size}) \not\simeq 0) \land \text{good-state-of-size}\,(\mathit{state},\ \mathit{size}))$
$\rightarrow \quad (\text{col-with-at-least}\,(\mathit{state},\ n,\ \mathit{size}) < \text{length}\,(\mathit{state}))$

THEOREM: lessp-length
$(n < \text{length}\,(\mathit{state})) \rightarrow \text{listp}\,(\mathit{state})$

EVENT: Disable equal-bitp-simplify.

THEOREM: listp-nat-to-bv
$\text{listp}\,(\text{nat-to-bv}\,(n,\ \mathit{size})) = (\mathit{size} \not\simeq 0)$

THEOREM: number-with-at-least-simple
$(\text{good-state-of-size}\,(\mathit{state},\ \mathit{size}) \land (\mathit{size} \simeq 0))$
$\rightarrow \quad (\text{number-with-at-least}\,(\mathit{state},\ n,\ \mathit{size}) = \text{length}\,(\mathit{state}))$

THEOREM: bitp-car
$\text{bvp}\,(bv) \rightarrow \text{bitp}\,(\text{car}\,(bv))$

THEOREM: car-xor-bv-better
$\text{car}\,(\text{xor-bv}\,(x,\ y))$
$= \quad$ **if** $\text{listp}\,(x) \land \text{listp}\,(y)$ **then** $\text{xor}\,(\text{car}\,(x),\ \text{car}\,(y))$
else 0 **endif**

THEOREM: xor-bv-inverse-2
$\text{xor-bv}\,(b,\ \text{xor-bv}\,(a,\ a)) = \text{firstn}\,(\text{fix-xor-bv}\,(b),\ \min\,(\text{length}\,(a),\ \text{length}\,(b)))$

THEOREM: length-fix-xor-bv
$\text{length}\,(\text{fix-xor-bv}\,(x)) = \text{length}\,(x)$

THEOREM: xor-bvs-put
$$(\text{good-state-of-size}\,(\mathit{state},\ \mathit{size}) \wedge (\text{length}\,(\mathit{value}) = \mathit{size}) \wedge (\mathit{place} < \text{length}\,(\mathit{state}))) \rightarrow (\text{xor-bvs}\,(\text{put}\,(\mathit{place},\ \mathit{value},\ \mathit{state})) = \text{xor-bv}\,(\text{get}\,(\mathit{place},\ \mathit{state}),\ \text{xor-bv}\,(\mathit{value},\ \text{xor-bvs}\,(\mathit{state}))))$$

DEFINITION:
triple-cdr-induction $(a,\ b,\ c)$
$=$ **if** listp $(a) \wedge$ listp $(b) \wedge$ listp (c)
then triple-cdr-induction (cdr (a), cdr (b), cdr (c))
else t endif

THEOREM: all-zeros-xor-bv-identity
$$((\text{length}\,(a) = \text{length}\,(b)) \wedge (\text{length}\,(b) = \text{length}\,(c)) \wedge \text{all-zeros}\,(c)) \rightarrow (\text{all-zeros}\,(\text{xor-bv}\,(a,\ \text{xor-bv}\,(b,\ c))) = (\text{fix-xor-bv}\,(a) = \text{fix-xor-bv}\,(b)))$$

THEOREM: length-get
$$(\text{good-state-of-size}\,(\mathit{state},\ \mathit{size}) \wedge (\mathit{place} < \text{length}\,(\mathit{state}))) \rightarrow (\text{length}\,(\text{get}\,(\mathit{place},\ \mathit{state})) = \text{fix}\,(\mathit{size}))$$

THEOREM: all-zeros-xor-bvs-put
$$(\text{all-zeros}\,(\text{xor-bvs}\,(\mathit{state})) \wedge \text{good-state-of-size}\,(\mathit{state},\ \mathit{size}) \wedge (\text{length}\,(\mathit{value}) = \mathit{size}) \wedge (\mathit{place} < \text{length}\,(\mathit{state}))) \rightarrow (\text{all-zeros}\,(\text{xor-bvs}\,(\text{put}\,(\mathit{place},\ \mathit{value},\ \mathit{state}))) = (\text{fix-xor-bv}\,(\mathit{value}) = \text{get}\,(\mathit{place},\ \mathit{state})))$$

THEOREM: lessp-bv-x-x
$$\neg\ \text{lessp-bv}\,(x,\ x)$$

THEOREM: put-get
$$(x < \text{length}\,(\mathit{state})) \rightarrow (\text{put}\,(x,\ \text{get}\,(x,\ \mathit{state}),\ \mathit{state}) = \mathit{state})$$

THEOREM: get-means-number-with-at-least
$$((\neg\ \text{lessp-bv}\,(\text{get}\,(x,\ \mathit{state}),\ \text{nat-to-bv}\,(n,\ \mathit{size}))) \wedge (x < \text{length}\,(\mathit{state}))) \rightarrow (\text{number-with-at-least}\,(\mathit{state},\ n,\ \mathit{size}) \neq 0)$$

THEOREM: number-with-at-least-put
$$((p < \text{length}\,(\mathit{state})) \wedge \text{good-state-of-size}\,(\mathit{state},\ \mathit{size}) \wedge (\text{length}\,(v) = \mathit{size})) \rightarrow (\text{number-with-at-least}\,(\text{put}\,(p,\ v,\ \mathit{state}),\ n,\ \mathit{size})$$

$$
\begin{array}{ll}
= & \textbf{if } \text{lessp-bv}\,(\text{get}\,(p,\ \mathit{state}),\ \text{nat-to-bv}\,(n,\ \mathit{size})) \\
& \textbf{then if } \text{lessp-bv}\,(v,\ \text{nat-to-bv}\,(n,\ \mathit{size})) \\
& \qquad \textbf{then } \text{number-with-at-least}\,(\mathit{state},\ n,\ \mathit{size}) \\
& \qquad \textbf{else } 1 + \text{number-with-at-least}\,(\mathit{state},\ n,\ \mathit{size})\ \textbf{endif} \\
& \textbf{elseif } \text{lessp-bv}\,(v,\ \text{nat-to-bv}\,(n,\ \mathit{size})) \\
& \textbf{then } \text{number-with-at-least}\,(\mathit{state},\ n,\ \mathit{size}) - 1 \\
& \textbf{else } \text{number-with-at-least}\,(\mathit{state},\ n,\ \mathit{size})\ \textbf{endif})
\end{array}
$$

THEOREM: all-zeros-xor-bvs-when-lone-big-simple
$(\text{good-state-of-size}\,(\mathit{state},\ \mathtt{1}) \wedge (\text{number-with-at-least}\,(\mathit{state},\ \mathtt{1},\ \mathtt{1}) = \mathtt{1}))$
$\rightarrow\quad (\text{car}\,(\text{xor-bvs}\,(\mathit{state})) = \mathtt{1})$

THEOREM: plus-exp-2-x-exp-2-x
$(\exp(\mathtt{2},\ x) + \exp(\mathtt{2},\ x)) = \exp(\mathtt{2},\ 1 + x)$

THEOREM: lessp-exp-exp
$(\exp(x,\ y) < \exp(x,\ z))$
$= \quad \textbf{if } x \simeq \mathtt{0}\ \textbf{ then } (y \not\simeq \mathtt{0}) \wedge (z \simeq \mathtt{0})$
$\qquad \textbf{else } (x \neq \mathtt{1}) \wedge (y < z)\ \textbf{endif}$

THEOREM: nat-to-bv-exp
$(n \not< \mathit{size}) \rightarrow (\text{nat-to-bv}\,(\exp(\mathtt{2},\ n),\ \mathit{size}) = \text{all-ones}\,(\mathit{size}))$

THEOREM: bv-to-nat-all-zeros
$\text{bvp}\,(\mathit{bv}) \rightarrow ((\text{bv-to-nat}\,(\mathit{bv}) = \mathtt{0}) = \text{all-zeros}\,(\mathit{bv}))$

THEOREM: lessp-bv-to-nat-1
$\text{bvp}\,(\mathit{bv}) \rightarrow ((\text{bv-to-nat}\,(\mathit{bv}) < \mathtt{1}) = \text{all-zeros}\,(\mathit{bv}))$

THEOREM: car-nat-to-bv-exp
$(n < \mathit{size})$
$\rightarrow\quad (\text{car}\,(\text{nat-to-bv}\,(\exp(\mathtt{2},\ n),\ \mathit{size}))$
$\qquad = \quad \textbf{if } (1 + n) = \mathit{size}\ \textbf{ then } \mathtt{1}$
$\qquad\qquad \textbf{else } \mathtt{0}\ \textbf{endif})$

```
;; let's disable some time wasters
```

EVENT: Disable equal-bitp-simplify.

EVENT: Disable all-zeros-xor-bvs-when-lone-big-simple.

EVENT: Disable high-bit-on-reasonable.

EVENT: Disable find-high-out-of-sync-reasonable.

THEOREM: all-zeros-firstn-difference
$(\text{bvp}(x) \land (\text{length}(x) = \textit{size}) \land (n < \textit{size}))$
$\rightarrow \quad (\text{lessp-bv}(x, \text{nat-to-bv}(\text{exp}(2, n), \textit{size}))$
$\quad = \quad \text{all-zeros}(\text{firstn}(x, \textit{size} - n)))$

THEOREM: all-zeros-xor-bv
$(\text{bvp}(a) \land \text{bvp}(b) \land (\text{length}(a) = \text{length}(b)))$
$\rightarrow \quad (\text{all-zeros}(\text{xor-bv}(a, b)) = (a = b))$

THEOREM: lessp-bv-nat-to-bv-1
$((\text{length}(x) = \textit{size}) \land \text{bvp}(x))$
$\rightarrow \quad (\text{lessp-bv}(x, \text{nat-to-bv}(1, \textit{size})) = ((0 < \textit{size}) \land \text{all-zeros}(x)))$

DEFINITION:
double-length-induction (x, y)
$=$ **if** listp $(x) \land$ listp (y) **then** double-length-induction (cdr (x), cdr (y))
else t endif

THEOREM: all-zeros-equal
$(\text{bvp}(a) \land \text{bvp}(b) \land \text{all-zeros}(a) \land \text{all-zeros}(b))$
$\rightarrow \quad ((a = b) = (\text{length}(a) = \text{length}(b)))$

EVENT: Disable all-zeros-equal.

THEOREM: all-zeros-xor-bvs
$(\text{good-state-of-size}(z, \textit{size}) \land (\text{number-with-at-least}(z, 1, \textit{size}) = 0))$
$\rightarrow \quad \text{all-zeros}(\text{xor-bvs}(z))$

THEOREM: length-firstn
$\text{length}(\text{firstn}(\textit{list}, \textit{size})) = \min(\text{length}(\textit{list}), \textit{size})$

THEOREM: length-xor-bv
$\text{length}(\text{xor-bv}(a, b)) = \min(\text{length}(a), \text{length}(b))$

THEOREM: firstn-xor-bv
$\text{firstn}(\text{xor-bv}(a, b), \textit{size}) = \text{xor-bv}(\text{firstn}(a, \textit{size}), \text{firstn}(b, \textit{size}))$

THEOREM: all-zeros-xor-bvs-firstn
$(\text{good-state-of-size}(z, \textit{size})$
$\land \quad (n < \textit{size})$
$\land \quad (\text{number-with-at-least}(z, \text{exp}(2, n), \textit{size}) = 0))$
$\rightarrow \quad \text{all-zeros}(\text{firstn}(\text{xor-bvs}(z), \textit{size} - n))$

THEOREM: all-zeros-if-firstn-zeros
$\text{all-zeros}(x) \rightarrow \text{all-zeros}(\text{firstn}(x, \textit{size}))$

THEOREM: good-state-of-size-length-xor-bvs
good-state-of-size $(z,\ \text{length}\,(\text{xor-bvs}\,(z)))$
$=\quad$ good-state-of-size $(z,\ \text{length}\,(\text{car}\,(z)))$

THEOREM: bvp-firstn
$\text{bvp}\,(x) \rightarrow \text{bvp}\,(\text{firstn}\,(x,\ \mathit{size}))$

THEOREM: number-with-at-least-exp-2
$(\text{good-state-of-size}\,(\mathit{state},\ \mathit{size})$
$\wedge\quad (n < \mathit{size})$
$\wedge\quad \text{all-zeros}\,(\text{firstn}\,(\text{xor-bvs}\,(\mathit{state}),\ \mathit{size} - n)))$
$\rightarrow\quad (\text{number-with-at-least}\,(\mathit{state},\ \exp\,(2,\ n),\ \mathit{size}) \neq 1)$

THEOREM: number-with-at-least-2
$(\text{good-state-of-size}\,(\mathit{state},\ \mathit{size})$
$\wedge\quad (1 < \mathit{size})$
$\wedge\quad \text{all-zeros}\,(\text{xor-bvs}\,(\mathit{state})))$
$\rightarrow\quad (\text{number-with-at-least}\,(\mathit{state},\ 2,\ \mathit{size}) \neq 1)$

THEOREM: lessp-bv-all-zeros
$(\text{all-zeros}\,(x) \wedge (\text{length}\,(x) = \text{length}\,(y)) \wedge \text{bvp}\,(x) \wedge \text{bvp}\,(y))$
$\rightarrow\quad ((\text{lessp-bv}\,(x,\ y) = (\neg\ \text{all-zeros}\,(y))) \wedge (\neg\ \text{lessp-bv}\,(y,\ x)))$

THEOREM: number-with-at-least-means-get
$(\text{good-state-of-size}\,(\mathit{state},\ \mathit{size})$
$\wedge\quad (z < \text{length}\,(\mathit{state}))$
$\wedge\quad (\text{number-with-at-least}\,(\mathit{state},\ n,\ \mathit{size}) = 0))$
$\rightarrow\quad ((\text{bv-to-nat}\,(\text{get}\,(z,\ \mathit{state})) < n) = \mathbf{t})$

THEOREM: remainder-sub1-hack
$((x\ \mathbf{mod}\ y) = p)$
$\rightarrow\quad (((x - 1)\ \mathbf{mod}\ y)$
$\quad = \quad \mathbf{if}\ x \simeq 0\ \ \mathbf{then}\ 0$
$\quad\quad\quad \mathbf{elseif}\ p \simeq 0\ \ \mathbf{then}\ y - 1$
$\quad\quad\quad \mathbf{else}\ p - 1\ \mathbf{endif})$

```
;; ought to be using more up-to-date naturals library!
```

THEOREM: equal-times-x
$((y * x) = x) = ((x = 0) \vee ((x \in \mathbf{N}) \wedge (y = 1)))$

THEOREM: equal-times-x-2
$((x * y) = x) = ((x = 0) \vee ((x \in \mathbf{N}) \wedge (y = 1)))$

THEOREM: equal-exp-x-x
$(\exp(x,\ y) = x)$
$= \quad ((x = \texttt{1}) \lor ((x = \texttt{0}) \land (y \not\simeq \texttt{0})) \lor ((x \in \mathbf{N}) \land (y = \texttt{1})))$

THEOREM: lessp-bv-2-means
$(\text{bvp}(x) \land (\texttt{1} < \text{length}(x)))$
$\to \quad (\text{lessp-bv}(x,\ \text{nat-to-bv}(\texttt{2},\ \text{length}(x)))$
$\qquad = \quad (\text{all-zeros}(x) \lor (x = \text{nat-to-bv}(\texttt{1},\ \text{length}(x)))))$

THEOREM: lessp-bv-x-1-means
$((\text{length}(x) = \mathit{size}) \land \text{bvp}(x) \land (\texttt{1} < \mathit{size}))$
$\to \quad (\text{lessp-bv}(x,\ \text{nat-to-bv}(\texttt{1},\ \mathit{size})) = \text{all-zeros}(x))$

THEOREM: nat-to-bv-not-numberp
$(n \notin \mathbf{N}) \to (\text{nat-to-bv}(n,\ \mathit{size}) = \text{nat-to-bv}(\texttt{0},\ \mathit{size}))$

THEOREM: all-zeros-nat-to-bv-1
$(\mathit{size} \not\simeq \texttt{0}) \to (\text{all-zeros}(\text{nat-to-bv}(n,\ \mathit{size})) = (n \simeq \texttt{0}))$

THEOREM: get-when-lessp-bv-2
$(\text{good-state-of-size}(\mathit{state},\ \mathit{size})$
$\land \quad (\texttt{1} < \mathit{size})$
$\land \quad (z < \text{length}(\mathit{state}))$
$\land \quad (\text{number-with-at-least}(\mathit{state},\ \texttt{2},\ \mathit{size}) = \texttt{0})$
$\land \quad \text{bvp}(x)$
$\land \quad (\text{length}(x) = \mathit{size}))$
$\to \quad (\text{lessp-bv}(x,\ \text{get}(z,\ \mathit{state}))$
$\qquad = \quad (\text{all-zeros}(x) \land (\text{get}(z,\ \mathit{state}) = \text{nat-to-bv}(\texttt{1},\ \mathit{size}))))$

```
;; replacement rule version
```

THEOREM: find-high-out-of-sync-reasonable2
$(\text{listp}(\mathit{state}) \land \text{good-state-of-size}(\mathit{state},\ \mathit{size}))$
$\to \quad ((\text{find-high-out-of-sync}(\mathit{state}) < \text{length}(\mathit{state})) = \mathbf{t})$

THEOREM: xor-bv-0
$((\text{length}(x) = \mathit{size}) \land \text{bvp}(x))$
$\to \quad ((\text{xor-bv}(\text{nat-to-bv}(\texttt{0},\ \mathit{size}),\ x) = x)$
$\qquad \land \quad (\text{xor-bv}(x,\ \text{nat-to-bv}(\texttt{0},\ \mathit{size})) = x))$

THEOREM: lessp-bv-to-nat-get-col-with-at-least
$(\text{good-state-of-size}(\mathit{state},\ \mathit{size})$
$\land \quad (n < \exp(\texttt{2},\ \mathit{size}))$
$\land \quad (\texttt{1} < \mathit{size})$
$\land \quad (\text{number-with-at-least}(\mathit{state},\ n,\ \mathit{size}) \not\simeq \texttt{0}))$
$\to \quad ((\text{bv-to-nat}(\text{get}(\text{col-with-at-least}(\mathit{state},\ n,\ \mathit{size}),\ \mathit{state})) < n) = \mathbf{f})$

```
;; needed?
```

THEOREM: lessp-get-exp-2-size
$$(\text{good-state-of-size}\,(\mathit{state},\ \mathit{size}) \land (z < \text{length}\,(\mathit{state})))$$
$$\rightarrow \quad ((\text{bv-to-nat}\,(\text{get}\,(z,\ \mathit{state})) < \exp\,(2,\ \mathit{size})) = \mathbf{t})$$

THEOREM: equal-remainder-sub1-0
$$(((x - 1)\ \mathbf{mod}\ p) = 0) = ((x \simeq 0) \lor (p = 1) \lor ((x\ \mathbf{mod}\ p) = 1))$$

THEOREM: lessp-1-rewrite
$$(1 < x) = ((x \in \mathbf{N}) \land (x \neq 0) \land (x \neq 1))$$

THEOREM: numberp-col-with-at-least
$$(\text{col-with-at-least}\,(\mathit{state},\ n,\ \mathit{size}) \in \mathbf{N}) = \text{listp}\,(\mathit{state})$$

THEOREM: equal-remainder-2-special
$$((((x\ \mathbf{mod}\ 2) = y) \land (y = 1)) \rightarrow ((x\ \mathbf{mod}\ 2) \neq 0))$$
$$\land \quad ((((x\ \mathbf{mod}\ 2) \neq y) \land (y = 1)) \rightarrow (((x\ \mathbf{mod}\ 2) = 0) = \mathbf{t}))$$

THEOREM: listp-fix-xor-bv
$$\text{listp}\,(\text{fix-xor-bv}\,(x)) = \text{listp}\,(x)$$

THEOREM: lessp-length-x-length-xor-bvs-put-x
$$(\text{length}\,(x) < \text{length}\,(\text{xor-bvs}\,(\text{put}\,(z,\ x,\ \mathit{state})))) = \mathbf{f}$$

THEOREM: length-xor-bvs-put
$$(\text{good-state-of-size}\,(\mathit{state},\ \mathit{size})$$
$$\land \quad \text{listp}\,(\mathit{state})$$
$$\land \quad (z < \text{length}\,(\mathit{state}))$$
$$\land \quad (\text{length}\,(x) = \mathit{size}))$$
$$\rightarrow \quad (\text{length}\,(\text{xor-bvs}\,(\text{put}\,(z,\ x,\ \mathit{state}))) = \mathit{size})$$

THEOREM: lessp-1-length
$$((n < \text{length}\,(x)) \land (n \not\simeq 0)) \rightarrow \text{listp}\,(\text{cdr}\,(x))$$

THEOREM: all-zeros-get-col-with-at-least
$$(\text{good-state-of-size}\,(\mathit{state},\ \mathit{size})$$
$$\land \quad (1 < \mathit{size})$$
$$\land \quad (1 < \text{length}\,(\mathit{state}))$$
$$\land \quad (n \not\simeq 0)$$
$$\land \quad (\text{number-with-at-least}\,(\mathit{state},\ n,\ \mathit{size}) \not\simeq 0))$$
$$\rightarrow \quad (\neg\ \text{all-zeros}\,(\text{get}\,(\text{col-with-at-least}\,(\mathit{state},\ n,\ \mathit{size}),\ \mathit{state})))$$

```
;; these dinosaurs aren't needed any more
```

EVENT: Disable equal-bit-1.

EVENT: Disable silly-listp-cdr.

```
;; this takes too much time and isn't needed often
```

EVENT: Disable nat-to-bv-is-all-ones.

THEOREM: equal-remainder-sub1-x-2-special
$(((x - 1) \textbf{ mod } 2) = 1) = ((x \not\simeq 0) \land ((x \textbf{ mod } 2) = 0))$

```
;;;;;;
;;;;;;
;; The big lemmas about NIM
```

THEOREM: smart-moves-from-not-loser
$((\neg \text{loser-state}(state, size))$
$\land \quad \text{good-state-of-size}(state, size)$
$\land \quad (1 < \text{length}(state))$
$\land \quad (1 < size)$
$\land \quad (\text{number-with-at-least}(state, 1, size) \neq 0))$
$\rightarrow \quad \text{loser-state}(\text{apply-move}(\text{smart-move}(state, size), state), size)$

DEFINITION:
stupid-move $(state, size)$
$= \quad$ **let** *pile* **be** col-with-at-least $(state, 1, size)$
in
cons $(pile, \text{nat-to-bv}(\text{bv-to-nat}(\text{get}(pile, state)) - 1, size))$ **endlet**

DEFINITION:
computer-move $(state, size)$
$= \quad$ **if** loser-state $(state, size)$ **then** stupid-move $(state, size)$
else smart-move $(state, size)$ **endif**

THEOREM: smart-move-is-a-move
$(\text{good-state-of-size}(state, size)$
$\land \quad (\neg \text{loser-state}(state, size))$
$\land \quad (1 < size)$
$\land \quad (1 < \text{length}(state))$
$\land \quad (0 < \text{number-with-at-least}(state, 1, size)))$
$\rightarrow \quad \text{movep}(\text{smart-move}(state, size), state, size)$

THEOREM: moves-from-loser
(loser-state (*state*, *size*)
 $\wedge$ $(\mathtt{1} < \textit{size})$
 $\wedge$ good-state-of-size (*state*, *size*)
 $\wedge$ movep (*move*, *state*, *size*))
$\rightarrow$ ($\neg$ loser-state (apply-move (*move*, *state*), *size*))

```
;;;; put together big lemmas in one theorem about "game"
```

CONSERVATIVE AXIOM: a-move-intro
((number-with-at-least (*state*, 1, *size*) $\not\simeq$ 0)
 $\wedge$ $(\mathtt{1} < \textit{size})$
 $\wedge$ good-state-of-size (*state*, *size*))
$\rightarrow$ movep (a-move (*state*, *size*), *state*, *size*)

Simultaneously, we introduce the new function symbol *a-move*.

DEFINITION:
reasonable-game-statep (*state*, *size*)
= (good-state-of-size (*state*, *size*)
 $\wedge$ $(\mathtt{1} < \textit{size})$
 $\wedge$ (1 < length (*state*)))

DEFINITION:
sum-matches (*state*)
= **if** listp (*state*)
 then bv-to-nat (car (*state*)) + sum-matches (cdr (*state*))
 else 0 **endif**

THEOREM: equal-x-nat-to-bv-0
(bvp (x) $\wedge$ all-zeros (x)) $\rightarrow$ ((x = nat-to-bv (0, length (x))) = **t**)

THEOREM: get-from-zeros
(good-state-of-size (z, *size*)
 $\wedge$ (*size* $\not\simeq$ 0)
 $\wedge$ (number-with-at-least (z, 1, *size*) = 0)
 $\wedge$ (d < length (z)))
$\rightarrow$ (get (d, z) = nat-to-bv (0, *size*))

THEOREM: lessp-sum-matches-member
(*place* < length (*state*))
$\rightarrow$ (sum-matches (*state*) $\not<$ bv-to-nat (get (*place*, *state*)))

THEOREM: sum-matches-put

$$
\begin{array}{l}
(\mathit{place} < \text{length}\,(\mathit{state})) \\
\rightarrow \quad (\text{sum-matches}\,(\text{put}\,(\mathit{place},\ v,\ \mathit{state})) \\
\qquad = \quad ((\text{sum-matches}\,(\mathit{state}) - \text{bv-to-nat}\,(\text{get}\,(\mathit{place},\ \mathit{state}))) \\
\qquad\qquad + \quad \text{bv-to-nat}\,(v)))
\end{array}
$$

THEOREM: lessp-not-all-zeros

$$((\neg\ \text{all-zeros}\,(x)) \wedge \text{bvp}\,(x) \wedge \text{listp}\,(x)) \rightarrow (\mathtt{0} < \text{bv-to-nat}\,(x))$$

DEFINITION:

$$
\begin{array}{l}
\text{game-ends-recursion}\,(\mathit{move},\ \mathit{state},\ \mathit{size}) \\
= \quad \textbf{if}\ \text{listp}\,(\mathit{state}) \wedge (\text{car}\,(\mathit{move}) \not\simeq \mathtt{0}) \\
\quad\ \ \textbf{then}\ \text{game-ends-recursion}\,(\text{cons}\,(\text{car}\,(\mathit{move}) - 1,\ \text{cdr}\,(\mathit{move})), \\
\qquad\qquad\qquad\qquad\qquad\quad \text{cdr}\,(\mathit{state}), \\
\qquad\qquad\qquad\qquad\qquad\quad \mathit{size}) \\
\quad\ \ \textbf{else}\ \mathbf{t}\ \textbf{endif}
\end{array}
$$

THEOREM: game-ends

$$
\begin{array}{l}
((\text{number-with-at-least}\,(\mathit{state},\ \mathtt{1},\ \mathit{size}) \neq \mathtt{0}) \\
\ \wedge \quad \text{good-state-of-size}\,(\mathit{state},\ \mathit{size}) \\
\ \wedge \quad (\mathit{size} \not\simeq \mathtt{0}) \\
\ \wedge \quad \text{movep}\,(\mathit{move},\ \mathit{state},\ \mathit{size})) \\
\rightarrow \quad ((\text{sum-matches}\,(\text{apply-move}\,(\mathit{move},\ \mathit{state})) < \text{sum-matches}\,(\mathit{state})) = \mathbf{t})
\end{array}
$$

EVENT: Disable apply-move.

EVENT: Disable smart-move.

EVENT: Disable stupid-move.

DEFINITION:

$$
\begin{array}{l}
\text{game}\,(\mathit{good\text{-}player\text{-}turn},\ \mathit{state},\ \mathit{size}) \\
= \quad \textbf{if}\ \neg\ \text{reasonable-game-statep}\,(\mathit{state},\ \mathit{size})\ \ \textbf{then}\ \mathbf{f} \\
\quad\ \ \textbf{elseif}\ \text{number-with-at-least}\,(\mathit{state},\ \mathtt{1},\ \mathit{size}) = \mathtt{0} \\
\quad\ \ \textbf{then}\ \mathit{good\text{-}player\text{-}turn} \\
\quad\ \ \textbf{else let}\ \mathit{move}\ \ \textbf{be}\ \ \textbf{if}\ \mathit{good\text{-}player\text{-}turn} \\
\qquad\qquad\qquad\qquad\quad \textbf{then}\ \text{computer-move}\,(\mathit{state},\ \mathit{size}) \\
\qquad\qquad\qquad\qquad\quad \textbf{else}\ \text{a-move}\,(\mathit{state},\ \mathit{size})\ \textbf{endif} \\
\qquad\quad \textbf{in} \\
\qquad\quad \textbf{if}\ \neg\ \text{movep}\,(\mathit{move},\ \mathit{state},\ \mathit{size})\ \ \textbf{then}\ \mathbf{f} \\
\qquad\quad \textbf{else}\ \text{game}\,(\neg\ \mathit{good\text{-}player\text{-}turn}, \\
\qquad\qquad\qquad\quad \text{apply-move}\,(\mathit{move},\ \mathit{state}), \\
\qquad\qquad\qquad\quad \mathit{size})\ \textbf{endif}\ \textbf{endlet}\ \textbf{endif}
\end{array}
$$

THEOREM: transitivity-of-lessp-bv
(lessp-bv $(x,\ y)$
 $\land$ $(\neg$ lessp-bv $(z,\ y))$
 $\land$ (length (x) = length (y))
 $\land$ (length (y) = length (z))
 $\land$ bvp (x)
 $\land$ bvp (y)
 $\land$ bvp (z))
$\rightarrow$ lessp-bv $(x,\ z)$

THEOREM: nat-to-bv-size-1
nat-to-bv $(x,\ 1)$
= **if** $x \simeq 0$ **then** '(0)
else '(1) **endif**

THEOREM: lessp-bv-nat-to-bv-nat-to-bv-2
$(x \not< y) \rightarrow (\neg$ lessp-bv (nat-to-bv $(x,\ size)$, nat-to-bv $(y,\ size)))$

THEOREM: lessp-bv-of-larger
(lessp-bv (v, nat-to-bv $(x,\ a)$) $\land$ (length $(v) = a$) $\land$ bvp (v) $\land$ $(y \not< x)$)
$\rightarrow$ lessp-bv (v, nat-to-bv $(y,\ a)$)

THEOREM: lessp-number-with-at-least-x-y
(($y \not< x$) $\land$ good-state-of-size ($state$, $size$))
$\rightarrow$ (number-with-at-least ($state$, x, $size$)
$\not<$ number-with-at-least ($state$, y, $size$))

THEOREM: number-with-at-least-x-y
((number-with-at-least ($state$, x, $size$) = 0)
 $\land$ $(y \not< x)$
 $\land$ good-state-of-size ($state$, $size$))
$\rightarrow$ (number-with-at-least ($state$, y, $size$) = 0)

THEOREM: listp-cdr-put
listp (cdr ($state$)) $\rightarrow$ listp (cdr (put (x, v, $state$)))

THEOREM: listp-cdr-apply-move
(listp (cdr ($state$)) $\land$ listp ($state$)) $\rightarrow$ listp (cdr (apply-move ($move$, $state$)))

THEOREM: good-state-of-size-apply-move
(movep ($move$, $state$, $size$) $\land$ good-state-of-size ($state$, $size$))
$\rightarrow$ good-state-of-size (apply-move ($move$, $state$), $size$)

```
;;;; THE GAME CORRECTNESS LEMMA
```

THEOREM: computer-always-wins
((*good-playerp* = (¬ loser-state (*state*, *size*)))
∧ reasonable-game-statep (*state*, *size*))
→ game (*good-playerp*, *state*, *size*)

```
;;;;;;
;;;;;;
;;; Let's run an example to watch the game.  We'll need to
;;; instantiate the dumb player's strategy to really do this,
;;; and we'll define some functions to relate bit vector states
;;; to integers.
```

DEFINITION:
nat-to-bv-state (*state*, *size*)
= **if** listp (*state*)
then cons (nat-to-bv (car (*state*), *size*), nat-to-bv-state (cdr (*state*), *size*))
else nil endif

DEFINITION:
bv-to-nat-state (*state*)
= **if** listp (*state*)
then cons (bv-to-nat (car (*state*)), bv-to-nat-state (cdr (*state*)))
else nil endif

```
;; a particular game where the other player uses the stupid strategy
```

DEFINITION:
game-with-stupid-move (*good-player-turn*, *state*, *size*)
= **if** ¬ reasonable-game-statep (*state*, *size*) **then f**
elseif number-with-at-least (*state*, `1`, *size*) = `0`
then *good-player-turn*
else let *move* **be if** *good-player-turn*
then computer-move (*state*, *size*)
else stupid-move (*state*, *size*) **endif**
in
if ¬ movep (*move*, *state*, *size*) **then f**
else game-with-stupid-move (¬ *good-player-turn*,
apply-move (*move*,
state),
size) **endif endlet endif**

THEOREM: movep-stupid-move
((number-with-at-least (*state*, `1`, *size*) ≠ `0`)
∧ (`1` < *size*)
∧ good-state-of-size (*state*, *size*))
→ movep (stupid-move (*state*, *size*), *state*, *size*)

Theorem: game-with-stupid-is-game

$$((\mathit{good\text{-}playerp} = (\neg\ \text{loser-state}\,(\mathit{state},\ \mathit{size})))$$
$$\land\quad \text{reasonable-game-statep}\,(\mathit{state},\ \mathit{size}))$$
$$\to\quad \text{game-with-stupid-move}\,(\mathit{good\text{-}playerp},\ \mathit{state},\ \mathit{size})$$

Index