Nim game proof Matt Wilding November 1991

; Requires the naturals library.

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

---

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 \( \text{XOR-BV}(x,y) \) be the bitwise exclusive or of \( b(x) \) and \( b(y) \). Let \( \text{XOR-BVS}(x_0, x_1, \ldots, x_n) \) be \( \text{XOR-BV}(x_0, \text{XOR-BV}(x_1, \ldots)) \).

A state is a LOSER state if

\[
\text{XOR-BVS}(p(0),\ldots,p(n)) = b(0) \text{ and there is an } i \text{ such that } P(i)>1 \text{ or }
\text{XOR-BVS}(p(0),\ldots,p(n)) = b(1) \text{ and there is no } i \text{ such that } P(i)>1.
\]

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))
```

3
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:**

\[
\text{put}(\text{place}, \text{value}, \text{state}) = \begin{cases} 
\text{cons}(\text{value}, \text{cdr}(\text{state})) & \text{if } \text{place} \approx 0 \\
\text{cons}(\text{car}(\text{state}), \text{put}(\text{place} - 1, \text{value}, \text{cdr}(\text{state}))) & \text{else}
\end{cases}
\]

**Definition:**

\[
\text{get}(\text{place}, \text{state}) = \begin{cases} 
\text{car}(\text{state}) & \text{if } \text{place} \approx 0 \\
\text{get}(\text{place} - 1, \text{cdr}(\text{state})) & \text{else}
\end{cases}
\]

**Theorem: get-put**

\[
\text{get}(a_1, \text{put}(a_2, \text{value}, \text{state})) = \begin{cases} 
\text{value} & \text{if } \text{fix}(a_1) = \text{fix}(a_2) \\
\text{get}(a_1, \text{state}) & \text{else}
\end{cases}
\]

**Definition:**

\[
\text{length}(\text{list}) = \begin{cases} 
1 + \text{length}(\text{cdr}(\text{list})) & \text{if } \text{listp}(\text{list}) \\
0 & \text{else}
\end{cases}
\]

**Event:** Introduce the function symbol \textit{bv-size} of 0 arguments.

**Definition:**

\[
\text{bitp}(\text{bit}) = ((\text{bit} = 0) \lor (\text{bit} = 1))
\]

**Definition:**

\[
\text{bvp}(\text{bv}) = \begin{cases} 
\text{bitp}(\text{car}(\text{bv})) \land \text{bvp}(\text{cdr}(\text{bv})) & \text{if } \text{listp}(\text{bv}) \\
\text{bv} = \text{nil} & \text{else}
\end{cases}
\]

4
DEFINITION:
bvsp \( (bvs) \) 
\[ = \begin{cases} 
\text{if listp (bvs) then } & \text{bvp (car (bvs)) } \land \text{bvsp (cdr (bvs))} \\
\text{else } & bvs = \text{nil} 
\end{cases} \] 

DEFINITION:
good-state-of-size \( (state, size) \) 
\[ = \begin{cases} 
\text{if listp (state) then } & \text{bvp (car (state)) } \land \text{length (car (state)) = fix (size)} \\
\land & \text{good-state-of-size (cdr (state), size)} \\
\text{else } & state = \text{nil} 
\end{cases} \] 

DEFINITION:
good-state \( (state) \) = good-state-of-size \( (state, BV\text{-SIZE}) \) 

DEFINITION:
lessp-bv \( (bv1, bv2) \) 
\[ = \begin{cases} 
\text{if listp (bv1) } \land \text{listp (bv2)} then & (\text{car (bv1)} < \text{car (bv2)}) \\
\lor & ((\text{car (bv1)} = \text{car (bv2)}) \land \text{lessp-bv (cdr (bv1), cdr (bv2))}) \\
\text{else } & \text{f endif} 
\end{cases} \]

; ; high order bit first 

DEFINITION:
nat-to-bv \( (nat, size) \) 
\[ = \begin{cases} 
\text{if } & \text{size } \simeq 0 \text{ then nil} \\
\text{elseif } & \text{nat } < \exp (2, size - 1) \text{ then cons (0, nat-to-bv (nat, size - 1))} \\
\text{else } & \text{cons (1, nat-to-bv (nat } - \exp (2, size - 1), size - 1)) \text{ endif} 
\end{cases} \]

; ; most significant bit first 

DEFINITION:
bv-to-nat \( (bv) \) 
\[ = \begin{cases} 
\text{if listp (bv) then } & \text{car (bv) } \ast \exp (2, \text{length (cdr (bv))) } + \text{bv-to-nat (cdr (bv))} \\
\text{else } & \text{0 endif} 
\end{cases} \]

THEOREM: length-nat-to-bv 
\[ \text{length (nat-to-bv (nat, size)) } = \text{fix (size)} \] 

THEOREM: bv-to-nat-nat-to-bv 
\[ \text{bv-to-nat (nat-to-bv (nat, size)) } = \begin{cases} 
\text{if } & \text{nat } < \exp (2, size) \text{ then fix (nat)} \\
\text{else } & \exp (2, size) - 1 \text{ endif} 
\end{cases} \]
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
\( ((\text{length}(x) = \text{length}(y)) \land \text{bvp}(x) \land \text{bvp}(y)) \)
\( \rightarrow ((\text{bv-to-nat}(x) < \text{bv-to-nat}(y)) = \text{lessp-bv}(x, y)) \)

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

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

Definition:
number-with-at-least \((state, min, size)\)
\[ = \begin{cases} 
\text{if listp}(state) & \\
\quad \text{then if } \neg \text{lessp-bv}(\text{car}(state), \text{nat-to-bv}(min, size)) & \\
\quad \quad \text{then } 1 + \text{number-with-at-least}(\text{cdr}(state), min, size) & \\
\quad \quad \text{else } \text{number-with-at-least}(\text{cdr}(state), min, size) \text{ endif} & \\
\quad \text{else } 0 \text{ endif} &
\end{cases} \]

;; return a column number with value at least min

Definition:
col-with-at-least \((state, min, size)\)
\[ = \begin{cases} 
\text{if listp}(state) & \\
\quad \text{then if } \neg \text{lessp-bv}(\text{car}(state), \text{nat-to-bv}(min, size)) & \\
\quad \quad \text{then } 0 & \\
\quad \quad \text{else } 1 + \text{col-with-at-least}(\text{cdr}(state), min, size) \text{ endif} & \\
\quad \text{else f endif} &
\end{cases} \]

Definition:
xor \((bit1, bit2)\)
\[ = \begin{cases} 
\text{if } (bit1 \simeq 0) = (bit2 \simeq 0) & \text{then } 0 & \\
\text{else } 1 \text{ endif} &
\end{cases} \]

Definition:
xor-bv \((bv1, bv2)\)
\[ = \begin{cases} 
\text{if listp}(bv1) \land \text{listp}(bv2) & \\
\quad \text{then cons}(\text{xor}(\text{car}(bv1), \text{car}(bv2)), \text{xor-bv}(\text{cdr}(bv1), \text{cdr}(bv2))) & \\
\quad \text{else nil endif} &
\end{cases} \]

Definition:
fix-bit \((x)\)
\[ = \begin{cases} 
\text{if } x \simeq 0 & \text{then } 0 & \\
\text{else } 1 \text{ endif} &
\end{cases} \]
**Definition**

\[ \text{fix-xor-bv}(bv) = \begin{cases} \text{nil} & \text{if listp}(bv) \text{ then } \text{cons}(\text{fix-bit(car}(bv)), \text{fix-xor-bv(cdr}(bv)))) \\ \text{if listp(cdr}(bv)) \text{ then xor-bv(car}(bv), \text{xor-bv}(\text{cdr}(bv))) & \text{else fix-xor-bv(car}(bv)) \text{ endif} \\ \text{else nil endif} \end{cases} \]

**Definition**

\[ \text{xor-bvs}(bvs) = \begin{cases} \text{nil} & \text{if listp(bvs)} \text{ then if listp(cdr}(bvs)) \text{ then xor-bv(car}(bvs), \text{xor-bvs}(\text{cdr}(bvs))) \\ \text{else fix-xor-bv(car}(bvs)) \text{ endif} \\ \text{else nil endif} \end{cases} \]

**Theorem:** \( bvp\text{-fix-xor-bv} \)

\[ bvp(\text{fix-xor-bv}(x)) \]

**Theorem:** \( bvp\text{-fix-xor-bv-identity} \)

\[ bvp(x) \rightarrow (\text{fix-xor-bv}(x) = x) \]

**Theorem:** \( \text{commutativity-of-xor} \)

\[ \text{xor}(a, b) = \text{xor}(b, a) \]

**Theorem:** \( \text{associativity-of-xor} \)

\[ \text{xor}(\text{xor}(a, b), c) = \text{xor}(a, \text{xor}(b, c)) \]

**Theorem:** \( \text{commutativity2-of-xor} \)

\[ \text{xor}(c, \text{xor}(a, b)) = \text{xor}(a, \text{xor}(c, b)) \]

**Theorem:** \( \text{commutativity-of-xor-bv} \)

\[ \text{xor-bv}(a, b) = \text{xor-bv}(b, a) \]

**Theorem:** \( \text{associativity-of-xor-bv} \)

\[ \text{xor-bv}(\text{xor-bv}(a, b), c) = \text{xor-bv}(a, \text{xor-bv}(b, c)) \]

**Theorem:** \( \text{commutativity2-of-xor-bv} \)

\[ \text{xor-bv}(c, \text{xor-bv}(a, b)) = \text{xor-bv}(a, \text{xor-bv}(c, b)) \]

;; return number of first vector with high bit on

**Definition**

\[ \text{high-bit-on}(bvs) = \begin{cases} \text{nil} & \text{if listp}(bvs) \text{ then if caar}(bvs) = 1 \text{ then 0} \\ \text{else 1 + high-bit-on(cdr}(bvs)) \text{ endif} \\ \text{else f endif} \end{cases} \]
**Definition:**
delete-pile\( (\text{place}, \text{state}) \) =
\[
\begin{align*}
\text{if} & \ \neg \ \text{listp}(\text{state}) \ \text{then} \ \text{state} \\
\text{elseif} & \ \text{place} \ \sim \ 0 \ \text{then} \ \text{cdr}(\text{state}) \\
\text{else} & \ \text{cons}(\text{car}(\text{state}), \text{delete-pile}(\text{place} - 1, \text{cdr}(\text{state}))) \ \text{endif}
\end{align*}
\]

**Definition:**
delete-high-bits\( (\text{state}) \) =
\[
\begin{align*}
\text{if} & \ \text{listp}(\text{state}) \ \text{then} \ \text{cons}(\text{cdar}(\text{state}), \text{delete-high-bits}(\text{cdr}(\text{state}))) \\
\text{else} & \ \text{nil} \ \text{endif}
\end{align*}
\]

**Theorem:** find-high-out-of-sync-rewrite
\[
\text{listp}(\text{state}) \land \text{listp}(\text{car}(\text{state})) \rightarrow (\text{length}(\text{car}(\text{delete-high-bits}(\text{state}))) < \text{length}(\text{car}(\text{state})))
\]

**Definition:**
find-high-out-of-sync\( (\text{state}) \) =
\[
\begin{align*}
\text{if} & \ \text{listp}(\text{state}) \ \text{then} \ \text{if} \ \text{listp}(\text{car}(\text{state})) \ \text{then} \ \text{if} \ \text{car}(\text{xor-bvs}(\text{state})) = 1 \ \text{then} \ \text{high-bit-on}(\text{state}) \\
\text{else} & \ \text{find-high-out-of-sync}(\text{delete-high-bits}(\text{state})) \ \text{endif} \\
\text{else} & \ \text{f} \ \text{endif}
\end{align*}
\]

**Definition:**
smart-move\( (\text{state}, \text{size}) \) =
\[
\begin{align*}
\text{if} & \ \text{number-with-at-least}(\text{state}, 2, \text{size}) = 0 \\
\text{then} & \ \text{cons}(\text{col-with-at-least}(\text{state}, 1, \text{size}), \text{nat-to-bv}(0, \text{size})) \\
\text{elseif} & \ \text{number-with-at-least}(\text{state}, 2, \text{size}) = 1 \\
\text{then} & \ \text{cons}(\text{col-with-at-least}(\text{state}, 2, \text{size}), \\
\text{if} & \ (\text{number-with-at-least}(\text{state}, 1, \text{size}) \ \text{mod} \ 2) = 0 \ \text{then} \ \text{nat-to-bv}(0, \text{size}) \\
\text{else} & \ \text{nat-to-bv}(1, \text{size}) \ \text{endif} \\
\text{else} & \ \text{let} \ \text{badcol} \ \text{be} \ \text{find-high-out-of-sync}(\text{state}) \ \text{in} \\
\text{cons}(\text{badcol}, \text{xor-bvs}(\text{delete-pile}(\text{badcol}, \text{state}))) \ \text{endlet} \ \text{endif}
\end{align*}
\]

**Definition:**
apply-move\( (\text{move}, \text{state}) = \text{put}(\text{car}(\text{move}), \text{cdr}(\text{move}), \text{state}) \)

**Definition:**
all-zeros\( (\text{bv}) \) =
\[
\begin{align*}
\text{if} & \ \text{listp}(\text{bv}) \ \text{then} \ (\text{car}(\text{bv}) = 0) \land \text{all-zeros}(\text{cdr}(\text{bv})) \\
\text{else} & \ \text{t} \ \text{endif}
\end{align*}
\]
**Definition:**
loser-state \((\text{state}, \text{size})\)
\[
= ((\text{number-with-at-least}(\text{state}, 2, \text{size}) = 0) \\
\quad \land ((\text{number-with-at-least}(\text{state}, 1, \text{size}) \mod 2) = 1)) \\
\lor ((0 < \text{number-with-at-least}(\text{state}, 2, \text{size})) \\
\quad \land \text{all-zeros} (\text{xor-bvs}(\text{state})))
\]

**Definition:**
movem \((\text{move}, \text{state}, \text{size})\)
\[
= (\text{lessp-bv}(\text{cdr}(\text{move}), \text{get}(\text{car}(\text{move}), \text{state})) \\
\quad \land (\text{length}(\text{cdr}(\text{move})) = \text{size}) \\
\quad \land \text{bvp}(\text{cdr}(\text{move})) \\
\quad \land (\text{car}(\text{move}) < \text{length}(\text{state})))
\]

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

**Definition:**
high-bit-out-of-sync \((\text{place}, \text{state})\)
\[
= \text{if } \text{listp}(\text{state}) \text{ then if } \text{listp}(\text{car}(\text{state})) \text{ then } ((\text{car}(\text{get}(\text{place}, \text{state})) = 1) \\
\quad \land (\text{car}(\text{xor-bvs}(\text{state})) = 1)) \\
\quad \lor ((\text{car}(\text{xor-bvs}(\text{state})) = 0) \\
\quad \land \text{high-bit-out-of-sync}(\text{place}, \text{delete-high-bits}(\text{state})))
\quad \text{else } f \text{ endif}
\quad \text{else } f \text{ endif}
\]

**Definition:**
lessp-when-high-bit-recursion \((\text{state}, \text{size})\)
\[
= \text{if } \text{size} \simeq 0 \text{ then } t \\
\quad \text{else } \text{lessp-when-high-bit-recursion}(\text{delete-high-bits}(\text{state}), \text{size} - 1) \text{ endif}
\]

**Theorem:** equal-length-0
\((\text{length}(\text{x}) = 0) = (\neg \text{listp}(\text{x}))\)

**Theorem:** equal-length-1
\((\text{length}(\text{x}) = 1) = (\text{listp}(\text{x}) \land (\neg \text{listp}(\text{cdr}(\text{x}))))\)

**Theorem:** good-state-of-size-delete-high-bits
\[
\text{good-state-of-size}(\text{state}, 1 + x) \\
\rightarrow \text{good-state-of-size}(\text{delete-high-bits}(\text{state}), x)
\]
Theorem: high-bit-out-of-sync-empty
\( \text{good-state-of-size} \left( \text{state}, \text{size} \right) \land \left( \text{size} \simeq 0 \right) \)
\( \rightarrow \left( \neg \text{high-bit-out-of-sync} \left( \text{place}, \text{state} \right) \right) \)

Theorem: get-of-bad-place
\( \text{place} \not< \text{length} \left( \text{state} \right) \rightarrow \left( \text{get} \left( \text{place}, \text{state} \right) = 0 \right) \)

Theorem: listp-delete-high-bits
\( \text{listp} \left( \text{delete-high-bits} \left( x \right) \right) = \text{listp} \left( x \right) \)

Definition:
\( \text{bv-not} \left( x \right) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{else} \end{cases} \)

Theorem: bvp-xor-bv
\( \text{bvp} \left( \text{xor-bv} \left( x, y \right) \right) \)

Theorem: bvp-xor-bvs
\( \text{bvp} \left( \text{xor-bvs} \left( \text{state} \right) \right) \)

Theorem: equal-bitp-simplify
\( \left( \text{bitp} \left( x \right) \land \left( x \not= 0 \right) \right) \rightarrow \left( \left( x = y \right) = \left( y = 1 \right) \right) \)

;; make all bit equality constants into 0

Theorem: equal-bit-1
\( \text{bitp} \left( x \right) \rightarrow \left( \left( x = 1 \right) = \left( x \not= 0 \right) \right) \)

Theorem: bitp-car-bvp
\( \text{bvp} \left( x \right) \rightarrow \text{bitp} \left( \text{car} \left( x \right) \right) \)

Theorem: good-state-of-size-means-bvsp
\( \text{good-state-of-size} \left( \text{state}, \text{size} \right) \rightarrow \text{bvsp} \left( \text{state} \right) \)

Theorem: listp-xor-bv
\( \text{listp} \left( \text{xor-bv} \left( x, y \right) \right) = \left( \text{listp} \left( x \right) \land \text{listp} \left( y \right) \right) \)

Theorem: listp-xor-bvs
\( \text{good-state-of-size} \left( \text{state}, \text{size} \right) \)
\( \rightarrow \left( \text{listp} \left( \text{xor-bvs} \left( \text{state} \right) \right) = \left( \text{listp} \left( \text{state} \right) \land \left( 0 < \text{size} \right) \right) \right) \)

Theorem: bvp-get
\( \text{good-state-of-size} \left( \text{state}, \text{size} \right) \)
\( \rightarrow \left( \text{bvp} \left( \text{get} \left( \text{place}, \text{state} \right) \right) = \left( \text{place} < \text{length} \left( \text{state} \right) \right) \right) \)
Theorem: listp-get
good-state-of-size (state, size)
→ (listp (get (place, state))
= ((place < length (state)) ∧ (0 < size)))

Theorem: car-xor-bv
(listp (x) ∧ listp (y)) → (car (xor-bv (x, y)) = xor (car (x), car (y)))

Definition:
lessp-bv-recursion (list, size)
= if size ≃ 0 then t
else lessp-bv-recursion (cdr (list), size − 1) endif

Definition:
firstn (list, n)
= if ¬ listp (list) then list
elseif n ≃ 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
xor-bv (a, xor-bv (a, b)) = firstn (fix-xor-bv (b), min (length (a), length (b)))

Theorem: xor-bv-fix-xor-bv
(xor-bv (fix-xor-bv (x), y) = xor-bv (x, y))
∧ (xor-bv (y, fix-xor-bv (x)) = xor-bv (y, x))

Theorem: firstn-noop
(x < length (list)) → (firstn (list, x) = list)

Theorem: xor-bvs-delete-high-bits
good-state-of-size (state, size)
→ (xor-bvs (delete-high-bits (state))
= if (¬ listp (state)) ∨ (1 < size) then nil
else cdr (xor-bvs (state)) endif)

Theorem: length-xor-bvs
good-state-of-size (state, size)
→ (length (xor-bvs (state))
= if listp (state) then fix (size)
else 0 endif)
Theorem: listp-delete-pile
\[
\text{listp}\ (\text{delete-pile}(\text{place}, \text{state})) = \left(\text{listp}\ (\text{state}) \land \neg \left(\left(\text{place} \simeq 0\right) \land \neg \text{listp}\ (\text{cdr}\ (\text{state}))\right)\right)
\]

Theorem: xor-bvs-delete-pile
\[
\begin{align*}
\text{(good-state-of-size}\ (\text{state}, \text{size}) \land \text{place} < \text{length}\ (\text{state})) & \rightarrow \text{(xor-bvs}\ (\text{delete-pile}\ (\text{place}, \text{state}))) \\
& = \text{if } 1 < \text{length}\ (\text{state}) \\
& \quad \text{then xor-bv}\ (\text{get}\ (\text{place}, \text{state}), \text{xor-bvs}\ (\text{state})) \\
& \quad \text{else nil endif}
\end{align*}
\]

Theorem: listp-delete-pile-delete-high-bits
\[
\text{listp}\ (\text{delete-pile}\ (\text{place}, \text{delete-high-bits}\ (\text{state}))) = \text{listp}\ (\text{delete-pile}\ (\text{place}, \text{state}))
\]

Theorem: get-delete-high-bits
\[
\text{get}\ (\text{place}, \text{delete-high-bits}\ (\text{state})) = \text{cdr}\ (\text{get}\ (\text{place}, \text{state}))
\]

Theorem: delete-pile-delete-high-bits
\[
\text{delete-pile}\ (\text{place}, \text{delete-high-bits}\ (\text{state})) = \text{delete-high-bits}\ (\text{delete-pile}\ (\text{place}, \text{state}))
\]

Theorem: good-state-of-size-delete-pile
\[
\text{good-state-of-size}\ (\text{state}, \text{size}) \rightarrow \text{good-state-of-size}\ (\text{delete-pile}\ (\text{place}, \text{state}), \text{size})
\]

Theorem: silly-listp-cdr
\[
(1 < \text{length}\ (x)) \rightarrow \text{listp}\ (\text{cdr}\ (x))
\]

Theorem: numberp-car-bv
\[
\text{bvp}\ (bv) \rightarrow \text{car}\ (bv) \in \mathbb{N}
\]

Theorem: length-delete-high-bits
\[
\text{length}\ (\text{delete-high-bits}\ (x)) = \text{length}\ (x)
\]

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

Theorem: xor-bvs-delete-high-bits-2
\[
\left((\text{size} \neq 0) \land \text{good-state-of-size}\ (\text{state}, \text{size})\right) \rightarrow \text{(xor-bvs}\ (\text{delete-high-bits}\ (\text{state}))) \\
& = \text{if } \neg \text{listp}\ (\text{state}) \lor (1 \not< \text{size}) \text{ then nil} \\
& \quad \text{else cdr}\ (\text{xor-bvs}\ (\text{state}))\ \text{endif}
\]

Theorem: lessp-when-high-bit-out-of-sync-helper
\[
\begin{align*}
\text{(good-state-of-size}\ (\text{state}, \text{size}) \land \text{place} < \text{length}\ (\text{state}) \land (1 < \text{length}\ (\text{state}) \land \text{high-bit-out-of-sync}\ (\text{place}, \text{state})) \rightarrow \text{lessp-bv}\ (\text{xor-bvs}\ (\text{delete-pile}\ (\text{place}, \text{state})), \text{get}\ (\text{place}, \text{state}))}
\end{align*}
\]

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-sync
\((\text{good-state-of-size}(state, size) \land (1 < \text{length}(state)) \land \text{high-bit-out-of-sync}(place, state)) \rightarrow \text{lessp-bv}(\text{xor-bvs}(\text{delete-pile}(place, state)), \text{get}(place, state))\)

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

Theorem: high-bit-on-works
\(((\text{car}(\text{xor-bvs}(state)) = 1) \land \text{bvsp}(\text{state})) \rightarrow (\text{car}(\text{get}(\text{high-bit-on}(state), state)) = 1)\)

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

Theorem: all-zeros-xor-bvs-simple
\((\text{good-state-of-size}(state, size) \land (size < 2)) \rightarrow (\text{all-zeros}(\text{xor-bvs}(state)) = ((\text{size} \approx 0) \lor (\neg \text{listp}(state)) \lor (\text{car}(\text{xor-bvs}(state)) = 0)))\)

Theorem: find-high-out-of-sync-works
\(((\neg \text{all-zeros}(\text{xor-bvs}(state))) \land \text{good-state-of-size}(state, size)) \rightarrow \text{high-bit-out-of-sync}(\text{find-high-out-of-sync}(state), state)\)

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

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

Definition:
\(\text{all-ones}(size) = \begin{cases} nil & \text{if } size \approx 0 \\ \text{cons}(1, \text{all-ones}(size - 1)) & \text{else} \end{cases}\)

Theorem: nat-to-bv-is-all-ones
\((\text{nat} \not< \exp(2, size)) \rightarrow (\text{nat-to-bv}(nat, size) = \text{all-ones}(size))\)
Theorem: length-all-ones
\[ \text{length (all-ones (size))} = \text{fix (size)} \]

Theorem: bvp-all-ones
\[ \text{bvp (all-ones (size))} \]

Theorem: lessp-bv-all-ones-arg1
\[ \text{bvp (bv)} \rightarrow (\neg \text{lessp-bv (all-ones (size), bv)}) \]

Theorem: lessp-bv-all-ones-arg2
\[ ((\text{length (bv)} = \text{size}) \land \text{bvp (bv)}) \rightarrow (\text{lessp-bv (bv, all-ones (size))}) = (\text{bv-to-nat (bv)} < (\exp(2, \text{size}) - 1))) \]

Theorem: lessp-bv-nat-to-bv
\[ ((\text{length (bv)} = \text{fix (size)}) \land (\text{nat} < \exp(2, \text{size}) \land \text{bvp (bv)})) \rightarrow ((\text{lessp-bv (nat-to-bv (nat, size), bv)} = (\text{nat} < \text{bv-to-nat (bv)})) \land (\text{lessp-bv (bv, nat-to-bv (nat, size)}) = (\text{bv-to-nat (bv)} < \text{nat}))) \]

Theorem: length-car-state
\[ \text{good-state-of-size (state, size)} \rightarrow (\text{length (car (state))}) = \begin{cases} \text{fix (size)} & \text{if listp (state)} \\ 0 & \text{else} \end{cases} \]

Theorem: bvp-car
\[ (\text{bvsp (x)} \land \text{listp (x)}) \rightarrow \text{bvp (car (x))} \]

Theorem: lessp-bv-zero
\[ (\text{zero} \simeq 0) \rightarrow (\neg \text{lessp-bv (x, nat-to-bv (zero, size)))} \]

Theorem: number-with-at-least-zero
\[ (\text{zero} \simeq 0) \rightarrow (\text{number-with-at-least (bv, zero, size}) = \text{length (bv)}) \]

Theorem: lessp-1-exp
\[ (1 < \exp (x, y)) = ((1 < x) \land (y \neq 0)) \]

Theorem: lessp-x-exp-x-y
\[ (x < \exp (x, y)) = (((x \simeq 0) \land (y \simeq 0)) \lor ((1 < x) \land (1 < y))) \]

Theorem: lessp-bv-col-get-with-at-least
\[ ((x < y) \land (y < \exp(2, \text{size}) \land \text{good-state-of-size (state, size)) \land (\text{number-with-at-least (state, y, size) \neq 0)})) \rightarrow \text{lessp-bv (nat-to-bv (x, size), get (col-with-at-least (state, y, size), state))} \]
THEOREM: lessp-1-x-means-not-zerop-x
\((1 < x) \rightarrow (x \not\equiv 0)\)

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

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

THEOREM: find-high-out-of-sync-reasonable
\(((\text{number-with-at-least}(\text{state}, n, \text{size}) \not\equiv 0) \land \text{good-state-of-size}(\text{state}, \text{size}))\) 
\(\rightarrow \ (\text{find-high-out-of-sync}(\text{state}) < \text{length}(\text{state}))\)

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

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

EVENT: Disable equal-bitp-simplify.

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

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

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

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

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

THEOREM: length-fix-xor-bv
\(\text{length}(\text{fix-xor-bv}(x)) = \text{length}(x)\)
**Theorem:** xor-bvs-put
\[
\begin{align*}
\text{good-state-of-size} & (\text{state}, \text{size}) \\
& \land (\text{length} (\text{value}) = \text{size}) \\
& \land (\text{place} < \text{length} (\text{state})) \\
\rightarrow & \ (\text{xor-bvs} (\text{put} (\text{place}, \text{value}, \text{state}))) \\
& = \ \text{xor-bv} (\text{get} (\text{place}, \text{state}), \text{xor-bv} (\text{value}, \text{xor-bvs} (\text{state})))
\end{align*}
\]

**Definition:**
\[
\text{triple-cdr-induction} (a, b, c) =
\begin{cases}
\text{triple-cdr-induction} (\text{cdr} (a), \text{cdr} (b), \text{cdr} (c)) & \text{if listp} (a) \land \text{listp} (b) \land \text{listp} (c) \\
t & \text{else}
\end{cases}
\]

**Theorem:** all-zeros-xor-bv-identity
\[
\begin{align*}
((\text{length} (a) = \text{length} (b)) \land (\text{length} (b) = \text{length} (c)) \land \text{all-zeros} (c)) \\
\rightarrow & \ (\text{all-zeros} (\text{xor-bv} (a, \text{xor-bv} (b, c)))) \\
& = \ (\text{fix-xor-bv} (a) = \text{fix-xor-bv} (b))
\end{align*}
\]

**Theorem:** length-get
\[
\begin{align*}
\text{good-state-of-size} (\text{state}, \text{size}) \land (\text{place} < \text{length} (\text{state})) \\
\rightarrow & \ (\text{length} (\text{get} (\text{place}, \text{state})) = \text{fix} (\text{size}))
\end{align*}
\]

**Theorem:** all-zeros-xor-bvs-put
\[
\begin{align*}
\text{all-zeros} (\text{xor-bvs} (\text{state})) \\
& \land \text{good-state-of-size} (\text{state}, \text{size}) \\
& \land (\text{length} (\text{value}) = \text{size}) \\
& \land (\text{place} < \text{length} (\text{state})) \\
\rightarrow & \ (\text{all-zeros} (\text{xor-bvs} (\text{put} (\text{place}, \text{value}, \text{state})))) \\
& = \ (\text{fix-xor-bv} (\text{value}) = \text{get} (\text{place}, \text{state}))
\end{align*}
\]

**Theorem:** lessp-bv-x-x
\[
\neg \text{lessp-bv} (x, x)
\]

**Theorem:** put-get
\[
(x < \text{length} (\text{state})) \rightarrow (\text{put} (x, \text{get} (x, \text{state}), \text{state}) = \text{state})
\]

**Theorem:** get-means-number-with-at-least
\[
((\neg \text{lessp-bv} (\text{get} (x, \text{state}), \text{nat-to-bv} (n, \text{size}))) \land (x < \text{length} (\text{state}))) \\
\rightarrow & \ (\text{number-with-at-least} (\text{state}, n, \text{size}) \neq 0)
\]

**Theorem:** number-with-at-least-put
\[
\begin{align*}
(p < \text{length} (\text{state})) \\
& \land \text{good-state-of-size} (\text{state}, \text{size}) \\
& \land (\text{length} (v) = \text{size}) \\
\rightarrow & \ (\text{number-with-at-least} (\text{put} (p, v, \text{state}), n, \text{size})
\end{align*}
\]

16
= \text{if} \ \text{lessp-bv} \ (\text{get} \ (p, \ state), \ \text{nat-to-bv} \ (n, \ size)) \\
\quad \text{then if} \ \text{lessp-bv} \ (v, \ \text{nat-to-bv} \ (n, \ size)) \\
\quad \quad \text{then} \ \text{number-with-at-least} \ (state, \ n, \ size) \\
\quad \quad \text{else} \ 1 + \ \text{number-with-at-least} \ (state, \ n, \ size) \ \text{endif} \\
\quad \text{elseif} \ \text{lessp-bv} \ (v, \ \text{nat-to-bv} \ (n, \ size)) \\
\quad \text{then} \ \text{number-with-at-least} \ (state, \ n, \ size) - 1 \\
\quad \text{else} \ \text{number-with-at-least} \ (state, \ n, \ size) \ \text{endif} \\

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

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

\text{THEOREM: lessp-exp-exp} \\
\quad (\text{exp} \ (x, \ y) < \text{exp} \ (x, \ z)) \\
\quad = \ \text{if} \ x \simeq 0 \ \text{then} \ (y \not\simeq 0) \ \land \ (z \simeq 0) \\
\quad \quad \text{else} \ (x \not= 1) \ \land \ (y < z) \ \text{endif} \\

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

\text{THEOREM: bv-to-nat-all-zeros} \\
\quad \text{bvp} \ (bv) \rightarrow ((\text{bv-to-nat} \ (bv) = 0) = \text{all-zeros} \ (bv)) \\

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

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

;; \text{let's disable some time wasters} \\

\text{EVENT: Disable equal-bitp-simplify.} \\

\text{EVENT: Disable all-zeros-xor-bvs-when-lone-big-simple.} \\

\text{EVENT: Disable high-bit-on-reasonable.} \\

\text{EVENT: Disable find-high-out-of-sync-reasonable.}
THEOREM: all-zeros-firstn-difference
\( \text{bvp}(x) \land (\text{length}(x) = \text{size}) \land (n < \text{size}) \)
\[ \rightarrow \ (\text{lessp-bv}(x, \text{nat-to-bv}(\exp(2, n), \text{size})) \]
\[ = \ \text{all-zeros}(\text{firstn}(x, \text{size} - n)) \]

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

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

DEFINITION:
\[
\text{double-length-induction}(x, y) =
\begin{cases}
\text{if} \ & \text{listp}(x) \land \text{listp}(y) \then \text{double-length-induction}(\text{cdr}(x), \text{cdr}(y)) \\
\text{else} \ & \text{t} \end{cases}
\]

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

EVENT: Disable all-zeros-equal.

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

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

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

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

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

THEOREM: all-zeros-if-firstn-zeros
\[ \text{all-zeros}(x) \rightarrow \text{all-zeros}(\text{firstn}(x, \text{size})) \]
Theorem: good-state-of-size-length-xor-bvs
\[ \text{good-state-of-size}(z, \text{length}(\text{xor-bvs}(z))) = \text{good-state-of-size}(z, \text{length}(\text{car}(z))) \]

Theorem: bvp-firstn
\[ \text{bvp}(x) \rightarrow \text{bvp}(\text{firstn}(x, \text{size})) \]

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

Theorem: number-with-at-least-2
\( (\text{good-state-of-size}(\text{state}, \text{size}) \land (1 < \text{size}) \land \text{all-zeros}(\text{xor-bvs}(\text{state}))) \rightarrow (\text{number-with-at-least}(\text{state}, 2, \text{size}) \neq 1) \)

Theorem: lessp-bv-all-zeros
\( (\text{all-zeros}(x) \land (\text{length}(x) = \text{length}(y)) \land \text{bvp}(x) \land \text{bvp}(y)) \rightarrow ((\text{lessp-bv}(x, y) = (\neg \text{all-zeros}(y))) \land (\neg \text{lessp-bv}(y, x))) \)

Theorem: number-with-at-least-means-get
\( (\text{good-state-of-size}(\text{state}, \text{size}) \land (z < \text{length}(\text{state})) \land (\text{number-with-at-least}(\text{state}, n, \text{size}) = 0)) \rightarrow ((\text{bv-to-nat}(\text{get}(z, \text{state})) < n) = t) \)

Theorem: remainder-sub1-hack
\( ((x \mod y) = p) \rightarrow (((x - 1) \mod y) = \text{if } x \approx 0 \text{ then } 0 \text{ elseif } p \approx 0 \text{ then } y - 1 \text{ else } p - 1 \text{ endif}) \); ought to be using more up-to-date naturals library!

Theorem: equal-times-x
\( ((y \times x) = x) = ((x = 0) \lor ((x \in \mathbb{N}) \land (y = 1))) \)

Theorem: equal-times-x-2
\( ((x \times y) = x) = ((x = 0) \lor ((x \in \mathbb{N}) \land (y = 1))) \)
Theorem: equal-exp-x-x
\[ (\exp(x, y) = x) = ((x = 1) \lor ((x = 0) \land (y \neq 0)) \lor ((x \in \mathbb{N}) \land (y = 1))) \]

Theorem: lessp-bv-2-means
\[ (\text{bvp}(x) \land (1 < \text{length}(x))) \rightarrow (\text{lessp-bv}(x, \text{nat-to-bv}(2, \text{length}(x)))) = (\text{all-zeros}(x) \lor (x = \text{nat-to-bv}(1, \text{length}(x)))) \]

Theorem: lessp-bv-x-1-means
\[ ((\text{length}(x) = \text{size}) \land \text{bvp}(x) \land (1 < \text{size})) \rightarrow (\text{lessp-bv}(x, \text{nat-to-bv}(1, \text{size})) = \text{all-zeros}(x)) \]

Theorem: nat-to-bv-not-numberp
\[ (n \notin \mathbb{N}) \rightarrow (\text{nat-to-bv}(n, \text{size}) = \text{nat-to-bv}(0, \text{size})) \]

Theorem: all-zeros-nat-to-bv-1
\[ (\text{size} \neq 0) \rightarrow (\text{all-zeros}(\text{nat-to-bv}(n, \text{size})) = (n \simeq 0)) \]

Theorem: xor-bv-0
\[ ((\text{length}(x) = \text{size}) \land \text{bvp}(x)) \rightarrow ((\text{xor-bv}(\text{nat-to-bv}(0, \text{size}), x) = x) \land (\text{xor-bv}(x, \text{nat-to-bv}(0, \text{size})) = x)) \]

Theorem: lessp-bv-to-nat-get-col-with-at-least
\[ (\text{good-state-of-size}(\text{state}, \text{size}) \land (n < \exp(2, \text{size})) \land (1 < \text{size}) \land (\text{number-with-at-least}(\text{state}, n, \text{size}) \neq 0)) \rightarrow ((\text{bv-to-nat}(\text{get}(\text{col-with-at-least}(\text{state}, n, \text{size}), \text{state})) < n) = \text{f}) \]
Theorem: lessp-get-exp-2-size
\((\text{good-state-of-size } (\text{state}, \text{size}) \land (z < \text{length } (\text{state})))\)
\(\rightarrow \ (\text{bv-to-nat } (\text{get } (z, \text{state})) < \text{exp } (2, \text{size})) = t\)

Theorem: equal-remainder-sub1-0
\(((x - 1) \mod p) = 0\) = \(((x \simeq 0) \lor (p = 1) \lor ((x \mod p) = 1))\)

Theorem: lessp-1-rewrite
\((1 < x) = ((x \in \mathbb{N}) \land (x \not= 0) \land (x \not= 1))\)

Theorem: numberp-col-with-at-least
\((\text{col-with-at-least } (\text{state}, n, size) \in \mathbb{N}) = \text{listp } (\text{state})\)

Theorem: equal-remainder-2-special
\(((x \mod 2) = y) \land (y = 1)\) \(\rightarrow \ ((x \mod 2) \not= 0)\)
\(\land \ ((((x \mod 2) \not= y) \land (y = 1)) \rightarrow (((x \mod 2) = 0) = t))\)

Theorem: listp-fix-xor-bv
\(\text{listp } (\text{fix-xor-bv } (x)) = \text{listp } (x)\)

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

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

Theorem: lessp-1-length
\(((n < \text{length } (x)) \land (n \not= 0)) \rightarrow \text{listp } (\text{cdr } (x))\)

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

;;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) \mod 2) = 1\) = \((x \not\equiv 0) \land ((x \mod 2) = 0)\)

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

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

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

Definition: computer-move (\text{state}, \text{size}) \\
= \text{if} \text{loser-state} (\text{state}, \text{size}) \text{ then stupid-move} (\text{state}, \text{size}) \\
\text{else} \text{smart-move} (\text{state}, \text{size}) \text{ endif}

Theorem: smart-move-is-a-move \\
(\text{good-state-of-size} (\text{state}, \text{size}) \\
\land (\neg \text{loser-state} (\text{state}, \text{size})) \\
\land (1 < \text{size}) \\
\land (1 < \text{length} (\text{state})) \\
\land (0 < \text{number-with-at-least} (\text{state}, 1, \text{size})) \\
\rightarrow \text{movep} (\text{smart-move} (\text{state}, \text{size}), \text{state}, \text{size})
Theorem: moves-from-loser
(loser-state (state, size)
∧ (1 < size)
∧ good-state-of-size (state, size)
∧ movep (move, state, size))
→ (¬ 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) ≠ 0)
∧ (1 < size)
∧ good-state-of-size (state, size))
→ 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)
∧ (1 < size)
∧ (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) ∧ all-zeros (x)) → ((x = nat-to-bv (0, length (x))) = t)

Theorem: get-from-zeros
(good-state-of-size (z, size)
∧ (size ≠ 0)
∧ (number-with-at-least (z, 1, size) = 0)
∧ (d < length (z)))
→ (get (d, z) = nat-to-bv (0, size))

Theorem: lessp-sum-matches-member
(place < length (state))
→ (sum-matches (state) ≠ bv-to-nat (get (place, state)))
Theorem: sum-matches-put
\[ \text{place} < \text{length} (\text{state}) \rightarrow (\text{sum-matches} (\text{put} (\text{place}, \text{v}, \text{state})) \]
\[ = (\text{sum-matches} (\text{state}) - \text{bv-to-nat} (\text{get} (\text{place}, \text{state}))) \]
\[ + \text{bv-to-nat} (\text{v})) \]

Theorem: lessp-not-all-zeros
\[ (\neg \text{all-zeros} (x) \land \text{bvp} (x) \land \text{listp} (x)) \rightarrow (0 < \text{bv-to-nat} (x)) \]

Definition:
\[ \text{game-ends-recursion} (\text{move}, \text{state}, \text{size}) \]
\[ = \begin{cases} 
\text{game-ends-recursion} (\text{cons} (\text{car} (\text{move}) - 1, \text{cdr} (\text{move})), \text{cdr} (\text{state}), \text{size}) & \text{if listp} (\text{state}) \land (\text{car} (\text{move}) \neq 0) \\
\text{t} & \text{else} 
\end{cases} \]

Theorem: game-ends
\[ ((\text{number-with-at-least} (\text{state}, 1, \text{size}) \neq 0) \land \text{good-state-of-size} (\text{state}, \text{size}) \land (\text{size} \neq 0) \land \text{movep} (\text{move}, \text{state}, \text{size})) \rightarrow ((\text{sum-matches} (\text{apply-move} (\text{move}, \text{state}))) < \text{sum-matches} (\text{state}) = \text{t}) \]

Event: Disable apply-move.

Event: Disable smart-move.

Event: Disable stupid-move.

Definition:
\[ \text{game} (\text{good-player-turn}, \text{state}, \text{size}) \]
\[ = \begin{cases} 
\text{f} & \text{if } \neg \text{reasonable-game-statep} (\text{state}, \text{size}) \\
\text{if } \text{number-with-at-least} (\text{state}, 1, \text{size}) = 0 & \text{then } \text{good-player-turn} \\
\text{else if } \text{good-player-turn} & \text{then } \text{computer-move} (\text{state}, \text{size}) \\
\text{else let } \text{move} & \text{be if } \text{good-player-turn} \\
\text{then } \text{a-move} (\text{state}, \text{size}) & \text{else} \text{endif} \\
\text{in} & \text{if } \neg \text{movep} (\text{move}, \text{state}, \text{size}) \text{ then } \text{f} \\
\text{else} \text{ game} (\neg \text{good-player-turn}, & \text{ apply-move} (\text{move}, \text{state}), \text{size} \text{) endif let endif} 
\end{cases} \]
Theorem: transitivity-of-lessp-bv
\[
\begin{align*}
\text{lessp-bv}(x, y) \\
\land (\neg \text{lessp-bv}(z, y)) \\
\land (\text{length}(x) = \text{length}(y)) \\
\land (\text{length}(y) = \text{length}(z)) \\
\land \text{bvp}(x) \\
\land \text{bvp}(y) \\
\land \text{bvp}(z)) \\
\rightarrow \text{lessp-bv}(x, z)
\end{align*}
\]

Theorem: nat-to-bv-size-1
\[
\text{nat-to-bv}(x, 1) = \begin{cases} 
\text{if } x \simeq 0 \text{ then } \text{'}(0) \\
\text{else } \text{'}(1) \text{ endif}
\end{cases}
\]

Theorem: lessp-bv-nat-to-bv-nat-to-bv-2
\[
(x \not< y) \rightarrow (\neg \text{lessp-bv}(\text{nat-to-bv}(x, \text{size}), \text{nat-to-bv}(y, \text{size})))
\]

Theorem: lessp-bv-of-larger
\[
\text{lessp-bv}(v, \text{nat-to-bv}(x, a)) \land (\text{length}(v) = a) \land \text{bvp}(v) \land (y \not< x)) \\
\rightarrow \text{lessp-bv}(v, \text{nat-to-bv}(y, a))
\]

Theorem: lessp-number-with-at-least-x-y
\[
((y \not< x) \land \text{good-state-of-size}(\text{state}, \text{size})) \\
\rightarrow (\text{number-with-at-least}(\text{state}, x, \text{size}) \\
\not< \text{number-with-at-least}(\text{state}, y, \text{size})
\]

Theorem: number-with-at-least-x-y
\[
(\text{number-with-at-least}(\text{state}, x, \text{size}) = 0) \\
\land (y \not< x) \\
\land \text{good-state-of-size}(\text{state}, \text{size}) \\
\rightarrow (\text{number-with-at-least}(\text{state}, y, \text{size}) = 0)
\]

Theorem: listp-cdr-put
\[
\text{listp}(\text{cdr}(\text{state})) \rightarrow \text{listp}(\text{cdr}(\text{put}(x, v, \text{state})))
\]

Theorem: listp-cdr-apply-move
\[
(\text{listp}(\text{cdr}(\text{state})) \land \text{listp}(\text{state})) \rightarrow \text{listp}(\text{cdr}(\text{apply-move}(\text{move}, \text{state})))
\]

Theorem: good-state-of-size-apply-move
\[
(\text{movep}(\text{move}, \text{state}, \text{size}) \land \text{good-state-of-size}(\text{state}, \text{size})) \\
\rightarrow \text{good-state-of-size}(\text{apply-move}(\text{move}, \text{state}), \text{size})
\]

;;; THE GAME CORRECTNESS LEMMA
Theorem: computer-always-wins

\[(\text{good-playerp} = (\neg \text{loser-state}(\text{state}, \text{size}))) \wedge \text{reasonable-game-statep}(\text{state}, \text{size}) \rightarrow \text{game}(\text{good-playerp}, \text{state}, \text{size})\]

;; 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(\text{state}, \text{size})
= \begin{cases} 
\text{if listp(\text{state})} 
\quad \text{then cons(nat-to-bv(car(\text{state}), \text{size}), nat-to-bv-state(cdr(\text{state}), \text{size}))} 
\quad \text{else nil} 
\end{cases}

Definition:

bv-to-nat-state(\text{state})
= \begin{cases} 
\text{if listp(\text{state})} 
\quad \text{then cons(bv-to-nat(car(\text{state})), bv-to-nat-state(cdr(\text{state})))} 
\quad \text{else nil} 
\end{cases}

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

Definition:

game-with-stupid-move(\text{good-player-turn}, \text{state}, \text{size})
= \begin{cases} 
\text{if } \neg \text{reasonable-game-statep}(\text{state}, \text{size}) \text{ then } f 
\quad \text{elseif number-with-at-least(\text{state}, 1, \text{size}) = 0} 
\quad \text{then } \text{good-player-turn} 
\quad \text{else let move be if } \text{good-player-turn} 
\quad \quad \text{then computer-move(\text{state}, \text{size})} 
\quad \quad \text{else stupid-move(\text{state}, \text{size})} 
\quad \text{endif in} 
\quad \text{if } \neg \text{movep(move, state, size)} \text{ then } f 
\quad \text{else game-with-stupid-move(\neg \text{good-player-turn},} 
\quad \text{apply-move(move, state),} 
\quad \text{size)} \text{endif} 
\end{cases}

Theorem: movep-stupid-move

\[(\text{number-with-at-least}(\text{state}, 1, \text{size}) \neq 0) \wedge (1 < \text{size}) \wedge \text{good-state-of-size}(\text{state}, \text{size}) \rightarrow \text{movep(stupid-move(\text{state}, \text{size}), \text{state}, \text{size})}\]
THEOREM: game-with-stupid-is-game

\((\text{good-playerp} = (\neg \text{loser-state}(\text{state}, \text{size}))) \land \text{reasonable-game-statep}(\text{state}, \text{size})) \rightarrow \text{game-with-stupid-move}(\text{good-playerp}, \text{state}, \text{size})\)
Index

a-move, 23, 24
a-move-intro, 23
all-ones, 13, 14, 17
all-zeros, 8, 9, 13, 16–21, 23, 24
all-zeros-equal, 18
all-zeros-firstn-difference, 18
all-zeros-get-col-with-at-least, 21
all-zeros-if-firstn-zeros, 18
all-zeros-length-1, 13
all-zeros-nat-to-bv-1, 20
all-zeros-xor-bv, 18
all-zeros-xor-bv-identity, 16
all-zeros-xor-bvs, 18
all-zeros-xor-bvs-firstn, 18
all-zeros-xor-bvs-put, 16
all-zeros-xor-bvs-simple, 13
all-zeros-xor-bvs-when-lone-big
   -simple, 17
apply-move, 8, 22–26
associativity-of-xor, 7
associativity-of-xor-bv, 7
bitp, 4, 10, 13, 15
bitp-car, 15
bitp-car-bvp, 10
bitp-car-xor-bvs, 13
bv-not, 10
bv-size, 4, 5
bv-to-nat, 5, 6, 13, 14, 17, 19–24, 26
bv-to-nat-all-zeros, 17
bv-to-nat-nat-to-bv, 5
bv-to-nat-state, 26
bvp, 4–7, 9, 10, 12–15, 17–20, 23–25
bvp-all-ones, 14
bvp-car, 14
bvp-firstn, 19
bvp-fix-xor-bv, 7
bvp-fix-xor-bv-identity, 7
bvp-get, 10
bvp-nat-to-bv, 13
bvp-xor-bv, 10
bvp-xor-bvs, 10
bvsp, 5, 10, 13, 14
car-nat-to-bv-exp, 17
car-xor-bv, 11
car-xor-bv-better, 15
col-with-at-least, 6, 8, 14, 15, 20–22
col-with-at-least-reasonable, 15
commutativity-of-xor, 7
commutativity-of-xor-bv, 7
commutativity2-of-xor, 7
commutativity2-of-xor-bv, 7
computer-always-wins, 26
computer-move, 22, 24, 26
delete-high-bits, 8–12
delete-pile, 8, 12, 13, 15
delete-pile-delete-high-bits, 12
double-length-induction, 18
equal-bit-1, 10
equal-bitp-simplify, 10
equal-exp-x-x, 20
equal-length-0, 9
equal-length-1, 9
equal-remainder-2-special, 21
equal-remainder-sub1-0, 21
equal-remainder-sub1-x-2-special
   1, 22
equal-times-x, 19
equal-times-x-2, 19
equal-x-nat-to-bv-0, 23
exp, 5, 6, 13, 14, 17–21
find-high-out-of-sync, 8, 13, 15, 20
find-high-out-of-sync-reasonable, 15
   2, 20
find-high-out-of-sync-rewrite, 8
find-high-out-of-sync-works, 13
firstn, 11, 15, 18, 19
firstn-noop, 11
firstn-xor-bv, 18
fix-bit, 6, 7
fix-xor-bv, 7, 11, 15, 16, 21

get, 4, 9–14, 16, 17, 19–24
get-delete-high-bits, 12
get-from-zeros, 23
get-means-number-with-at-least, 16
get-put, 4
get-when-lessp-bv-2, 20
good-state, 5
good-state-of-size, 5, 9–26
good-state-of-size-apply-move, 25
good-state-of-size-delete-high-bits, 9
good-state-of-size-delete-pile, 12
good-state-of-size-length-xor-bvs, 19
good-state-of-size-means-bvsp, 10

high-bit-on, 7, 8, 13, 15
high-bit-on-reasonable, 15
high-bit-on-works, 13
high-bit-out-of-sync, 9, 10, 12, 13
high-bit-out-of-sync-empty, 10
high-bit-out-of-sync-trivial, 13

length, 4–6, 8–16, 18–25
length-all-ones, 14
length-car-state, 14
length-delete-high-bits, 12
length-firstn, 18
length-fix-xor-bv, 15
length-get, 16
length-nat-to-bv, 5
length-xor-bv, 18
length-xor-bvs, 11
length-xor-bvs-delete-pile, 15
length-xor-bvs-put, 21

lessp-1-exp, 14
lessp-1-length, 21
lessp-1-rewrite, 21
lessp-1-x-means-not-zerop-x, 15
lessp-bv, 5, 6, 9, 12–14, 16–20, 25
lessp-bv-2-means, 20
lessp-bv-all-ones-arg1, 14
lessp-bv-all-ones-arg2, 14
lessp-bv-all-zeros, 19
lessp-bv-col-get-with-at-least, 14
lessp-bv-length, 6
lessp-bv-nat-to-bv, 14
lessp-bv-nat-to-bv-1, 18
lessp-bv-nat-to-bv-nat-to-bv, 6
lessp-bv-nat-to-bv-nat-to-bv-2, 25
lessp-bv-of-larger, 25
lessp-bv-recursion, 11
lessp-bv-to-nat-1, 17
lessp-bv-to-nat-bv-to-nat, 6
lessp-bv-to-nat-get-col-with-at-least, 20
lessp-bv-x-1-means, 20
lessp-bv-x-x, 16
lessp-bv-zero, 14
lessp-exp-exp, 17
lessp-get-exp-2-size, 21
lessp-length, 15
lessp-length-x-length-xor-bvs-put-x, 21
lessp-not-all-zeros, 24
lessp-number-with-at-least-x-y, 25
lessp-sum-matches-member, 23
lessp-when-high-bit-out-of-sync, 13
-lessp-helper, 12
lessp-when-high-bit-recursion, 9
lessp-x-exp-x-y, 14
listp-cdr-apply-move, 25
listp-cdr-put, 25
listp-delete-high-bits, 10
listp-delete-pile, 12
listp-delete-pile-delete-high-bits, 12
listp-fix-xor-bv, 21
listp-get, 11
listp-nat-to-bv, 15
listp-xor-bv, 10
listp-xor-bvs, 10
loser-state, 9, 22, 23, 26, 27
min, 11, 15, 18
movep, 9, 22–26
movep-stupid-move, 26
moves-from-loser, 23

nat-to-bv, 5, 6, 8, 13–18, 20, 22, 23, 25, 26
nat-to-bv-exp, 17
nat-to-bv-is-all-ones, 13
nat-to-bv-not-numberp, 20
nat-to-bv-size-1, 25
nat-to-bv-state, 26
number-with-at-least, 6, 8, 9, 14–26
number-with-at-least-2, 19
number-with-at-least-exp-2, 19
number-with-at-least-means-get, 19
number-with-at-least-put, 16
number-with-at-least-simple, 15
number-with-at-least-x-y, 25
number-with-at-least-zero, 14
numberp-car-bv, 12
numberp-col-with-at-least, 21

plus-exp-2-x-exp-2-x, 17
put, 4, 8, 16, 21, 24, 25
put-get, 16

reasonable-game-statep, 23, 24, 26, 27
remainder-sub1-hack, 19

silly-lessp-sub1-exp-length, 13
silly-listp-cdr, 12
smart-move, 8, 22
smart-move-is-a-move, 22
smart-moves-from-not-loser, 22
stupid-move, 22, 26
sum-matches, 23, 24
sum-matches-put, 24

transitivity-of-lessp-bv, 25
triple-cdr-induction, 25

xor, 6, 7, 11, 15
xor-bv, 6, 7, 10–12, 15, 16, 18, 20
xor-bv-0, 20
xor-bv-fix-xor-bv, 11
xor-bv-inverse, 11
xor-bv-inverse-2, 15
xor-bvs, 7–13, 15–19, 21
xor-bvs-delete-high-bits, 11
xor-bvs-delete-high-bits-2, 12
xor-bvs-delete-pile, 12
xor-bvs-put, 16

30