Reference Manual for the

Elk Bit String Extension

Oliver Laumann

1. Introduction

The bit string extension to Elk defines a new data type *bitstring* (a sequence of zero or more bits) and functions to create and manipulate bit strings. The bits in a bit string are numbered beginning from zero up to the number of bits minus one; bit number 0 is the least significant bit (LSB), and the highest numbered bit is the most significant bit (MSB).

The print representation of bit strings is introduced by the sequence '#*'; the bits are printed starting with the most significant bit. Likewise, in the reader the sequence '#*' introduces a bit string constant.

Example:

```
#*0100110111
#*                  (empty bit string)
```

2. Using the Bit String Extension

To load the bit string extension, evaluate the expression

```
(require 'bitstring)
```

This causes the files *bitstring.scm* and *bitstring.o* to be loaded (*bitstring.o* must be statically linked with the interpreter on platforms that do not support dynamic loading of object files).

Loading the bit string extension causes the features *bitstring* and *bitstring.o* to be provided.

3. Creating Bit Strings

(make-bitstring *length init***)** **procedure**

make-bitstring returns a new bit string of the given length. If init is #t, all bits are initialized to 1; if init is #f, all bits are initialized to 0.

(bitstring-copy *bitstring***)** **procedure**

This procedure returns a copy of the specified bit string.

(bitstring-append *bitstring*$_1$ *bitstring*$_2$**)** **procedure**

bitstring-append returns a new bit string holding the concatenation of the specified bit string arguments.

4. Bit String Predicates

(bitstring? *obj***)** **procedure**

This type predicate returns #t if *obj* is a bit string, #f otherwise.

(bitstring=? *bitstring*$_1$ *bitstring*$_2$**)** **procedure**

This procedure returns #t if the bit string arguments are of the same length and contain the same bits, #f otherwise.

(bitstring-zero? *bitstring***)** **procedure**

bitstring-zero? returns #t if the specified bit string contains only 0 bits, #f otherwise.

5. Integer Conversions

(unsigned-integer–>bitstring *length i***)** **procedure**

(signed-integer–>bitstring *length i***)** **procedure**

Both procedures convert the exact integer argument *i* into a bit string of *length* bits and return the bit string. *length* must be large enough to hold the bit string representation of *i*. The integer argument to *unsigned-integer->bitstring* must be non-negative. *signed-integer->bitstring* uses two's complement representation for negative integers.

(bitstring–>unsigned-integer *bitstring***)** **procedure**

(bitstring–>signed-integer *bitstring***)** **procedure**

Both procedures convert the given bit string into an integer. *bitstring->signed-integer* interprets the bit string as the two's complement representation of a signed integer.

6. Selecting Components of Bit Strings

(bitstring-length *bitstring***)** **procedure**

This procedure returns the number of bits in the specified bit string.

(bitstring-ref *bitstring index***)** **procedure**

bitstring-ref returns #t if the *index*-th bit in the given bit string is 1, #f otherwise.

(bitstring-substring *bitstring from to***)** **procedure**

This procedure returns a new bit string initialized with the bits of *bitstring* starting at the index *from* (inclusive) and ending at the index *to* (exclusive).

7. Modifying Bit Strings

(bitstring-fill! *bitstring init***)** **procedure**

This procedure sets all bits in the specified bit string to 1 if *init* is #t, or to 0 if *init* is #f. It returns the non-printing object.

(bitstring-set! *bitstring index init***)** **procedure**

bitstring-set! sets the *index*-th bit in the specified bit string to 1 if *init* is #t, or to 0 if *init* is #f. It returns the non-printing object.

(bitstring-move! *dst-bitstring src-bitstring***)** **procedure**

bitstring-move! destructively copies the contents of the bit string *src-bitstring* into *dst-bitstring*. Both bit strings must have the same length. It returns the non-printing object.

(bitstring-substring-move! *src-bitstring* $from_1$ to_1 *dst-bitstring* $from_2$**)** **procedure**

This procedure destructively copies the bits from *src-bitstring* starting at index $from_1$ (inclusive) and ending at index to_1 (exclusive) into *dst-bitstring* starting at index $from_2$ (inclusive). Overlapping is handled correctly. The procedure returns the non-printing object.

8. Bitwise Logical Operations

(bitstring-not *bitstring***)** **procedure**

This procedure returns a new bit string initialized to the bitwise logical negation of the given bit string.

(bitstring-not! *dst-bitstring src-bitstring***)** **procedure**

This procedure destructively overwrites the contents of *dst-bitstring* with the bitwise logical negation of the bits in *src-bitstring*. Both bit strings must have the same length. *bitstring-not!* returns the non-printing object.

(bitstring-and $bitstring_1$ $bitstring_2$**)** **procedure**

(bitstring-andnot $bitstring_1$ $bitstring_2$**)** **procedure**
(bitstring-or $bitstring_1$ $bitstring_2$**)** **procedure**
(bitstring-xor $bitstring_1$ $bitstring_2$**)** **procedure**

These procedures return a new bit string initialized to the bitwise logical *and* (logical *and* with the negation, logical *or*, logical exclusive *or*, respectively) of the two bit string arguments. The two bit strings must have the same length.

(bitstring-and! *dst-bitstring src-bitstring***)** **procedure**

(bitstring-or! *dst-bitstring src-bitstring***)** **procedure**
(bitstring-andnot! *dst-bitstring src-bitstring***)** **procedure**
(bitstring-xor! *dst-bitstring src-bitstring***)** **procedure**

These procedures are the destructive versions of the four bitwise logical procedures described above. They perform the corresponding logical operation on the two bit string arguments and overwrite the contents of *dst-bitstring* with the result. Both bit strings must have the same length. These procedures return the non-printing object.

Index

U

Table of Contents