Moscow ML Library Documentation

Version 2.00 of June 2000

Sergei Romanenko, Russian Academy of Sciences, Moscow, Russia Claudio Russo, Cambridge University, Cambridge, United Kingdom Peter Sestoft, Royal Veterinary and Agricultural University, Copenhagen, Denmark

This document

This manual describes the Moscow ML library, which includes parts of the SML Basis Library and several extensions. The manual has been generated automatically from the commented signature files.

Alternative formats of this document

Hypertext on the World-Wide Web

The manual is available at http://www.dina.kvl.dk/~sestoft/mosmllib/ for online browsing.

Hypertext in the Moscow ML distribution

The manual is available for offline browsing at mosml/doc/mosmllib/index.html in the distribution.

On-line help in the Moscow ML interactive system

The manual is available also in interactive mosml sessions. Type help "lib"; for an overview of built-in function libraries. Type help "fromstring"; for help on a particular identifier, such as fromString. This will produce a menu of all library structures which contain the identifier fromstring (disregarding the lowercase/uppercase distinction):

1	val	Bool.fromString
2	val	Char.fromString
3	val	Date.fromString
4	val	Int.fromString
5	val	Path.fromString
6	val	Real.fromString
7	val	String.fromString
8	val	Time.fromString
9	val	Word.fromString
10	val	Word8.fromString

Choosing a number from this menu will invoke the help browser on the desired structure, e.g. Int.

The Moscow ML home page is $\label{lower} http://www.dina.kvl.dk/~sestoft/mosml.html$

CONTENTS 1

Contents		Redblackmap 116	
		Redblackset	117
AppleScript	2	Regex	119
Array	3	SML90	123
Array2	5	Signal	124
ArraySlice	8	Socket	126
Arraysort	11	Splaymap	131
BinIO	12	Splayset	132
Binarymap	15	String	134
Binaryset	16	StringCvt	137
Bool	18	Substring	139
Buffer	19	Susp	143
Byte	20	TextIO	144
Callback	21	Time	147
Char	24	Timer	149
CharArray	27	Unix	150
CharArraySlice	28	Vector	151
CharVector	29	VectorSlice	153
CharVectorSlice	30	Weak	155
CommandLine	31	Word	158
Date	32	Word8	161
Dynarray	35	Word8Array	164
Dynlib	36	Word8ArraySlice	165
FileSys	38	Word8Vector	166
Gdbm	41	Word8VectorSlice	167
Gdimage	43	Index	168
General	46	macx	100
Hashset	50		
Help	52		
Int	53		
Intmap	55		
Intset	56		
Lexing	58		
List	60		
ListPair	62		
Listsort	64		
Location	65		
Math	67		
	69		
Meta	73		
Mosml	73 74		
Mosmlegi	74 77		
Mosmlcookie			
Msp	78 84		
Mysql			
NJ93	88		
Nonstdio	90		
OS	91		
Option	92		
PP	93		
Parsing	96		
Path	98		
Polygdbm	101		
Polyhash	103		
Postgres	105		
Process	109		
Random	110		
Rbset	111		
Real	114		

2 APPLESCRIPT

Module AppleScript

val as_run_text

AppleScript -- Apple MacOS scripting

type OSAID

type OSAerr = int

exception AppleScriptErr of OSAerr * string

val as_compile : string -> OSAID

val as_dispose : OSAID -> unit

val as_run_script : OSAID -> string

: string -> string

These Mac specific functions provide the capability to compile and run AppleScript programs.

The exception AppleScriptErr is raised in the event of an error.

[as_compile str] compiles AppleScript source code text, returning an abstract token of type OSAID. This token may be used to run the script. The token may be used repeatedly until it is returned with as_dispose or until mosml exits.

[as_dispose tok] disposes of the resources associated with the OSAID token so that they may be reused by the AppleScript system. AppleScriptErr is raised upon any attemp to reuse a disposed token.

[as_run_script tok] runs the script associated with the token. This typically involves AppleEvent communication with other programs running on the Mac, or networked Macs. The AppleScript result is returned as a string.

[as_run_text str] compiles and runs the AppleScript source code text, disposing all resources allocated in the process, and returns the AppleScript result as a string.

References:

Inside Macintosh: Interapplication Communication, Chapter 10
AppleScript Language Guide English Edition,
available at http://applescript.apple.com/support.html

ARRAY 3

Module Array

prim_EQtype 'a array

Array -- SML Basis Library

Raises Subscript if i<0 or i>=length a.

starting at index di.

[copy{src, dst, di}] destructively copies the array src to dst,

Raises Subscript if di<0, or if $di+length\ src>length\ dst.$

```
val maxLen : int
val array : int * '_a -> '_a array
val tabulate : int * (int -> '_a) -> '_a array
val fromList : ' a list -> ' a array
val length
                            : 'a array -> int
                           : 'a array * int -> 'a
: 'a array * int * 'a -> unit
val sub
val update
val vector
                           : 'a array -> 'a Vector.vector
val copy : \{src: 'a array, dst: 'a array, di: int\} \rightarrow unit val copyVec : <math>\{src: 'a vector, dst: 'a array, di: int\} \rightarrow unit val copyVec : \{src: 'a vector, dst: 'a array, di: int\} \rightarrow unit val copyVec : \{src: 'a array, dst: 'a array, di: int\} \rightarrow unit val copyVec : \{src: 'a array, dst: 'a array, di: int\} \rightarrow unit val copyVec : \{src: 'a array, dst: 'a array, di: int\} \rightarrow unit val copyVec : \{src: 'a array, dst: 'a array
                            : ('a -> bool) -> 'a array -> 'a option
val find
val exists : ('a -> bool) -> 'a array -> bool
                            : ('a -> bool) -> 'a array -> bool
val all
                            : ('a -> unit) -> 'a array -> unit
val app
                           : ('a * 'b -> 'b) -> 'b -> 'a array -> 'b
: ('a * 'b -> 'b) -> 'b -> 'a array -> 'b
val foldl
val foldr
val modify : ('a -> 'a) -> 'a array -> unit
                           : (int * 'a -> bool) -> 'a array -> (int * 'a) option
: (int * 'a -> unit) -> 'a array -> unit
val findi
val appi
val foldli : (int * 'a * 'b -> 'b) -> 'b -> 'a array -> 'b
                             : (int * 'a * 'b -> 'b) -> 'b -> 'a array -> 'b
val foldri
val modifyi : (int * 'a -> 'a) -> 'a array -> unit
val collate : ('a * 'a -> order) -> 'a array * 'a array -> order
       ['ty array] is the type of one-dimensional, mutable, zero-based
       constant-time-access arrays with elements of type 'ty. Type
       'ty array admits equality even if 'ty does not. Arrays al and a2
       are equal if both were created by the same call to a primitive
       (array, tabulate, fromList).
       Functions working on a slices (contiguous subsequence) of an array
      are found in the ArraySlice structure.
       [maxLen] is the maximal number of elements in an array.
       [\operatorname{array}(n,\ x)] returns a new array of length n whose elements are all x.
      Raises Size if n<0 or n>maxLen.
       [\, \texttt{tabulate}(\, \texttt{n}, \,\, \texttt{f}\,) \,] \,\, \texttt{returns} \,\, \texttt{a} \,\, \texttt{new} \,\, \texttt{array} \,\, \texttt{of} \,\, \texttt{length} \,\, \texttt{n} \,\, \texttt{whose} \,\, \texttt{elements}
       are f 0, f 1, ..., f (n-1), created from left to right. Raises
      Size if n<0 or n>maxLen.
      [fromList xs] returns an array whose elements are those of xs. Raises Size if length xs > maxLen.
       [length a] returns the number of elements in a.
       [sub(a, i)] returns the i'th element of a, counting from 0.
      Raises Subscript if i<0 or i>=length a. To make 'sub' infix, use
       the declaration
                                                                 infix 9 sub
       [update(a, i, x)] destructively replaces the i'th element of a by x.
```

4 ARRAY

[copyVec{src, dst, di}] destructively copies the vector to dst, starting at index di. Raises Subscript if di<0, or if di + Vector.length src > length dst.

[find p a] applies p to each element x of a, from left to right, until p(x) evaluates to true; returns SOME x if such an x exists, otherwise NONE.

[exists p a] applies p to each element x of a, from left to right, until p(x) evaluates to true; returns true if such an x exists, otherwise false.

[all p a] applies p to each element x of a, from left to right, until p(x) evaluates to false; returns false if such an x exists, otherwise true.

[foldl f e a] folds function f over a from left to right. That is, computes $f(a[len-1], f(a[len-2], \ldots, f(a[1], f(a[0], e)) \ldots))$, where len is the length of a.

[foldr f e a] folds function f over a from right to left. That is, computes $f(a[0], f(a[1], \ldots, f(a[len-2], f(a[len-1], e)) \ldots))$, where len is the length of a.

[app f a] applies f to a[j] for j=0,1,..., length a-1.

[modify f a] applies f to a[j] and updates a[j] with the result f(a[j]) for j=0,1,...,length a-1.

The following iterators generalize the above ones by passing also the index j to the function being iterated.

[findi p a] applies f to successive pairs (j, a[j]) for j=0,1,...,n-1, until p(j, a[j]) evaluates to true; returns SOME (j, a[j]) if such a pair exists, otherwise NONE.

[foldli f e a] folds function f over the array from left to right. That is, computes f(n-1, a[n-1], f(..., f(1, a[1], f(0, a[0], e)) ...)).

[foldri f e a] folds function f over the array from right to left. That is, computes f(0, a[0], f(1, a[1], ..., f(n-1, a[n-1], e) ...)).

[appi f a] applies f to successive pairs (j, a[j]) for j=0,1,...,n-1.

[modifyi f a] applies f to (j, a[j]) and updates a[j] with the result f(j, a[j]) for j=0,1,...,n-1.

[collate cmp (xs, ys)] returns LESS, EQUAL or GREATER according as xs precedes, equals or follows ys in the lexicographic ordering on arrays induced by the ordering cmp on elements.

ARRAY2 5

Module Array2

```
Array2 -- SML Basis Library
eqtype 'a array
datatype traversal = RowMajor | ColMajor
val array : int * int * '_a -> '_a array
val fromList : '_a list list -> '_a array
val tabulate : traversal -> int * int * (int * int -> '_a) -> '_a array
val dimensions : 'a array -> int * int
val nCols : 'a array -> int
                  : 'a array -> int
val nRows
                 : 'a array * int * int -> 'a
: 'a array * int * int * 'a -> unit
val sub
val update
                  : 'a array * int -> 'a Vector.vector
                  : 'a array * int -> 'a Vector.vector
val column
: { src : 'a region, dst : 'a array,
val copy
                       dst_row : int, dst_col : int } -> unit
val app
                  : traversal -> ('a -> unit) -> 'a array -> unit
                  : traversal -> ('a -> 'a) -> 'a array -> unit
: traversal -> ('a * 'b -> 'b) -> 'b -> 'a array -> 'b
val modify
val fold
                 : traversal -> (int * int * 'a -> unit) -> 'a region -> unit
: traversal -> (int * int * 'a -> 'a) -> 'a region -> unit
: traversal -> (int * int * 'a * 'b -> 'b) -> 'b
val appi
val modifyi
val foldi
                    -> 'a region -> 'b
    ['ty array] is the type of two-dimensional, mutable, zero-based
```

['ty array] is the type of two-dimensional, mutable, zero-based constant-time-access arrays with elements of type 'ty.

Type 'ty array admits equality even if 'ty does not. Arrays al and a2 are equal if both were created by the same call to one of the primitives array, fromList, and tabulate.

[traversal] is the type of traversal orders: row major or column major.

[RowMajor] specifies that an operation must be done in row-major order, that is, one row at a time, from top to bottom, and from left to right within each row. Row-major traversal visits the elements of an (m,n)-array with m rows and n columns in this order:

```
(0,0), (0,1), (0,2), ..., (0,n-1), (1,0), (1,1), (1,2), ..., (1,n-1),
```

that is, in order of lexicographically increasing (i, j). In Moscow ML, row-major traversal is usually faster than column-major traversal.

[ColMajor] specifies that an operation must be done in column-major order, that is, one column at a time, from left to right, and from top to bottom within each column. Column-major traversal visits the elements of an (m,n)-array with m rows and n columns in this order:

that is, in order of lexicographically increasing (j, i).

[array(m, n, x)] returns a new m * n matrix whose elements are all x. Raises Size if n<0 or m<0.

[fromList xss] returns a new array whose first row has elements

6 ARRAY2

xs1, second row has elements xs2, ..., where xss = [xs1,xs2,...,xsm]. Raises Size if the lists in xss do not all have the same length.

[tabulate RowMajor (m, n, f)] returns a new m-by-n array whose elements are f(0,0), f(0,1), ..., f(0, n-1), f(1,0), f(1,1), ..., f(1, n-1),

 $f(m-1,0)\,,\qquad\ldots,\qquad f(m-1,\ n-1)$ created in row-major order: $f(0,0)\,,\,\,f(0,1)\,,\,\,\ldots,\,\,f(1,0)\,,\,\,f(1,1)\,,\,\,\ldots$ Raises Size if n<0 or m<0.

[tabulate ColMajor (m, n, f)] returns a new m-by-n array whose elements are as above, but created in the column-major order: f(0,0), f(1,0), ..., f(0,1), f(1,1), ... Raises Size if n<0 or m<0.

[dimensions a] returns the dimensions $(m,\ n)$ of a, where m is the number of rows and n the number of columns.

[nCols a] returns the number of n of columns of a.

[nRows a] returns the number of m of rows of a.

[sub(a, i, j)] returns the i'th row's j'th element, counting from 0. Raises Subscript if i<0 or j<0 or i>=m or j>=n where (m,n) = dimensions a.

[update(a, i, j, x)] destructively replaces the (i,j)'th element of a by x. Raises Subscript if i<0 or j<0 or i>=m or j>=n where (m,n) = dimensions a.

[row (a, i)] returns a vector containing the elements of the ith row of a. Raises Subscript if i < 0 or i >= height a.

[column (a, j)] returns a vector containing the elements of the jth column of a. Raises Subscript if j < 0 or j >= width a.

[app RowMajor f a] applies f to the elements a[0,0], a[0,1], ..., a[0,n-1], a[1,0], ..., a[m-1, n-1] of a, where (m, n) = dimensions a.

[app ColMajor f a] applies f to the elements a[0,0], a[1,0], ..., a[n-1,0], a[0,1], a[1,1], ..., a[m-1, n-1] of a, where (m, n) = dimensions a.

[modify RowMajor f a] applies f to the elements a[0,0], a[0,1], ..., a[0,n-1], a[1,0], ..., a[m-1, n-1] of a, updating each element with the result of the application, where (m, n) = dimensions a.

[modify ColMajor f a] applies f to the elements a[0,0], a[1,0], ..., a[n-1,0], a[0,1], a[1,1], ..., a[m-1, n-1] of a, updating each element with the result of the application, where $(m, n) = \dim n$ dimensions a.

[fold RowMajor f b a] folds f left-right and top-down over the elements of a in row-major order. That is, computes $f(a[m-1,\ n-1],\ f(a[m-1,\ n-2],\ \dots,\ f(a[0,1],\ f(a[0,0],\ b))\ \dots))$ where $(m,\ n)$ = dimensions a.

[fold ColMajor f b a] folds f left-right and top-down over the elements of a in column-major order. That is, computes $f(a[m-1,\ n-1],\ f(a[m-2,\ n-1],\ \dots,\ f(a[1,0],\ f(a[0,0],\ b))\ \dots))$ where $(m,\ n)$ = dimensions a.

The following iterators generalize the above ones in two ways:

- * the indexes i and j are also being passed to the function;
- * the iterators work on a region (submatrix) of a matrix.

[region] is the type of records { base, row, col, nrows, ncols }
determining the region or submatrix of array base whose upper left
corner has index (row, col).

ARRAY2

7

If nrows = SOME r, then the region has r rows: row, row+1, ..., row+r-1. If nrows = NONE, then the region extends to the bottom of the matrix. The field ncols similarly determines the number of columns.

A region is valid for an array with dimensions (m, n) if

(1) either nrows = NONE and 0 <= row <= m

or nrows = SOME r and 0 <= row <= row + r <= m and (2) either ncols = NONE and 0 <= col <= n or ncols = SOME c and 0 <= col <= col + c <= n.

[appi RowMajor f reg] applies f to (i, j, a[i, j]) in order of lexicographically increasing (i, j) within the region reg. Raises Subscript if reg is not valid. Note that app tr f a is equivalent to appi tr (f o #3) {base=a, row=0, col=0, nrows=NONE, ncols=NONE}

[appi ColMajor f reg] applies f to (i, j, a[i, j]) in order of lexicographically increasing (j, i) within the region reg. Raises Subscript if reg is not valid.

[modifyi RowMajor f reg)] applies f to (i, j, a[i, j]) in order of lexicographically increasing (i, j) within the region reg. Raises Subscript if reg is not valid. Note that modify tr f a is equivalent to modifyi (f o #3) {base=a, row=0, col=0, nrows=NONE, ncols=NONE}).

[modifyi ColMajor f reg)] applies f to (i, j, a[i, j]) in order of lexicographically increasing (j, i) within the region reg. Raises Subscript if reg is not valid.

[foldi RowMajor f b a] folds f over (i, j, a[i, j]) in row-major order within the region reg, that is, for lexicographically increasing (i, j) in the region. Raises Subscript if reg is not valid.

[foldi ColMajor f b a] folds f over (i, j, a[i, j]) in column-major order within the region reg, that is, for lexicographically increasing (j, i) in the region. Raises Subscript if reg is not valid.

[copy { src, dst, dst_row, dst_col }] copies the region determined by src to array dst such that the upper leftmost corner of src is copied to dst[dst_row, dst_col]. Works correctly even when src and dst are the same and the source and destination regions overlap. Raises Subscript if the src region is invalid, or if src translated to (dst_row, dst_col) is invalid for dst.

8 ARRAYSLICE

Module ArraySlice

```
ArraySlice -- SML Basis Library
type 'a slice
             : 'a slice -> int
val length
              : 'a slice * int -> 'a
              : 'a slice * int * 'a -> unit
: 'a Array.array * int * int option -> 'a slice
val update
val slice
             : 'a Array.array -> 'a slice
val full
val subslice : 'a slice * int * int option -> 'a slice
val base
              : 'a slice -> 'a Array.array * int * int
             : 'a slice -> 'a Vector.vector
val vector
             : {src: 'a slice, dst: 'a Array.array, di: int} -> unit
val copy
val copyVec : {src: 'a VectorSlice.slice, dst: 'a Array.array, di: int}
                  -> unit
val isEmpty : 'a slice -> bool
val getItem : 'a slice -> ('a * 'a slice) option
             : ('a -> bool) -> 'a slice -> 'a option
val find
val exists : ('a -> bool) -> 'a slice -> bool
              : ('a -> bool) -> 'a slice -> bool
val all
              : ('a -> unit) -> 'a slice -> unit
val app
             : ('a * 'b -> 'b) -> 'b -> 'a slice -> 'b
: ('a * 'b -> 'b) -> 'b -> 'a slice -> 'b
val foldl
val foldr
val modify : ('a \rightarrow 'a) \rightarrow 'a slice \rightarrow unit
              : (int * 'a -> bool) -> 'a slice -> (int * 'a) option
: (int * 'a -> unit) -> 'a slice -> unit
val findi
val appi
              : (int * 'a * 'b -> 'b) -> 'b -> 'a slice -> 'b
val foldli
val foldri
              : (int * 'a * 'b -> 'b) -> 'b -> 'a slice -> 'b
val modifyi : (int * 'a -> 'a) -> 'a slice -> unit
val collate : ('a * 'a -> order) -> 'a slice * 'a slice -> order
```

```
['ty slice] is the type of array slices, that is, sub-arrays.
The slice (a,i,n) is valid if 0 \le i \le i+n \le size s, or equivalently, 0 \le i and 0 \le n and i+n \le size s.
A valid slice sli = (a,i,n) represents the sub-array a[i...i+n-1],
so the elements of sli are a[i], a[i+1], ..., a[i+n-1], and n is
the length of the slice. Only valid slices can be constructed by
the functions below.
[length sli] returns the number n of elements in sli = (s,i,n).
[sub (sli, k)] returns the k'th element of the slice, that is,
a(i+k) where sli = (a,i,n). Raises Subscript if k<0 or k>=n.
[update (sli, k, x)] destructively replaces the k'th element of sli
by x. That is, replaces a(k+i) by x, where sli = (a,i,n). Raises
Subscript if i<0 or i>=n.
[slice (a, i, NONE)] creates the slice (a, i, length a-i),
consisting of the tail of a starting at i.
Raises Subscript if i<0 or i > Array.length a.
Equivalent to slice (a, i, SOME(Array.length a - i)).
[slice (a, i, SOME n)] creates the slice (a, i, n), consisting of
the sub-array of a with length n starting at i. Raises Subscript
if i<0 or n<0 or i+n > Array.length a.
```

slice	meaning			
(a, 0, NONE) (a, 0, SOME n) (a, i, NONE) (a, i, SOME n)	the whole array a left sub-array (prefix) a right sub-array (suffix) a general slice	a[0len-1] a[0n-1] a[ilen-1] a[ii+n-1]		

[full a] creates the slice (a, 0, length a).

Equivalent to slice(a,0,NONE)

[subslice (sli, i', NONE)] returns the slice (a, i+i', n-i') when sli = (a,i,n). Raises Subscript if i' < 0 or i' > n.

[subslice (sli, i', SOME n')] returns the slice (a, i+i', n') when sli = (a,i,n). Raises Subscript if i' < 0 or n' < 0 or i'+n' > n.

[base sli] is the concrete triple (a, i, n) when sli = (a, i, n).

[vector sli] creates and returns a vector consisting of the elements of the slice, that is, a[i..i+n-1] when sli = (a,i,n).

[copy $\{src, dst, di\}\]$ copies the elements of slice src = (a,i,n), that is, a[i..i+n-1], to the destination segment dst[di..di+n-1]. Raises Subscript if di<0 or if di+n> length dst. Works also if the array underlying sli is the same as dst, and the slice overlaps with the destination segment.

[copyVec {src, dst, di}] copies the elements of the vector slice src = (v,i,n), that is, v[i..i+n-1], to dst[di..di+n-1]. Raises Subscript if di<0, or if len=NONE and di + n > length dst.

[isEmpty sli] returns true if the slice sli = (a,i,n) is empty, that is, if n=0.

[getItem sli] returns SOME(x, rst) where x is the first element and rst the remainder of sli, if sli is non-empty; otherwise returns NONE.

[find p sli] applies p to each element x of sli, from left to right, until p(x) evaluates to true; returns SOME x if such an x exists, otherwise NONE.

[exists p sli] applies p to each element x of sli, from left to right, until p(x) evaluates to true; returns true if such an x exists, otherwise false.

[all p sli] applies p to each element x of sli, from left to right, until p(x) evaluates to false; returns false if such an x exists, otherwise true.

[app f sli] applies f to all elements of sli = (a,i,n), from left to right. That is, applies f to a[j+i] for $j=0,1,\ldots,n$.

[foldl f e sli] folds function f over sli = (a,i,n) from left to right. That is, computes f(a[i+n-1], f(a[i+n-2],..., f(a[i+1], f(a[i], e))...)).

[foldr f e sli] folds function f over sli = (a,i,n) from right to left. That is, computes f(a[i], f(a[i+1],..., f(a[i+n-2], f(a[i+n-1], e))...)).

[modify f sli] modifies the elements of the slice sli = (a,i,n) by function f. That is, applies f to a[i+j] and updates a[i+j] with the result f(a[i+j]) for $j=0,1,\ldots,n$.

The following iterators generalize the above ones by also passing the index into the array a underlying the slice to the function being iterated.

[findi p sli] applies p to the elements of sli = (a,i,n) and the underlying array indices, and returns the least (j, a[j]) for which p(j, a[j]) evaluates to true, if any; otherwise returns NONE. That is, evaluates p(j, a[j]) for j=i,..i+n-1 until it evaluates to true for some j, then returns SOME(j, a[j]); otherwise returns NONE.

[appi f sli] applies f to the slice sli = (a,i,n) and the underlying array indices. That is, applies f to successive pairs (j, a[j]) for j=i,i+1,...,i+n-1.

[foldli f e sli] folds function f over the slice sli = (a,i,n) and the underlying array indices from left to right. That is, computes f(i+n-1, a[i+n-1], f(..., f(i+1, a[i+1], f(i, a[i], e)) ...)).

10 ARRAYSLICE

[foldri f e sli] folds function f over the slice sli = (a,i,n) and the underlying array indices from right to left. That is, computes $f(i, a[i], f(i+1, a[i+1], \ldots, f(i+n-1, a[i+n-1], e) \ldots))$.

[modifyi f sli] modifies the elements of the slice sli = (a,i,n) by applying function f to the slice elements and the underlying array indices. That is, applies f to (j, a[j]) and updates a[j] with the result f(j, a[j]) for $j=i,i+1,\ldots,i+n-1$.

[collate cmp (sli1, sli2)] returns LESS, EQUAL or GREATER according as sli1 precedes, equals or follows sli2 in the lexicographic ordering on slices induced by the ordering cmp on elements.

ARRAYSORT 11

Module Arraysort

```
Arraysort -- Quicksort for arrays, from SML/NJ library
val sort : ('a * 'a -> order) -> 'a Array.array -> unit
val sorted : ('a * 'a -> order) -> 'a Array.array -> bool
```

[sort ordr arr] sorts array arr in-place, using ordering relation ordr.

[sorted ordr arr] returns true if the elements of array arr is appear in (weakly) increasing order, according to ordering ordr.

12 BINIO

Module BinIO

BinIO -- SML Basis Library type elem = Word8.word type vector = Word8Vector.vector Binary input type instream val openIn : string -> instream val closeIn : instream -> unit val input : instream -> vector val inputAll : instream -> vector val inputNoBlock : instream -> vector option
val input1 : instream -> elem option val inputN : instream * int -> vector val endOfStream : instream -> bool val lookahead : instream -> elem option Binary output type outstream val openOut : string -> outstream val openAppend : string -> outstream val closeOut : outstream -> unit val output : outstream * vector -> unit

This structure provides input/output functions on byte streams. The functions are state-based: reading from or writing to a stream changes the state of the stream. The streams are buffered: output to a stream may not immediately affect the underlying file or device.

[instream] is the type of state-based byte input streams.

: outstream * elem -> unit
: outstream -> unit

[outstream] is the type of state-based byte output streams.

[elem] is the type Word8.word of bytes.

[vector] is the type of Word8Vector.vector (byte vectors).

BYTE INPUT:

val output1 val flushOut

[openIn s] creates a new instream associated with the file named s. Raises Io.Io is file s does not exist or is not accessible.

[closeIn istr] closes stream istr. Has no effect if istr is closed already. Further operations on istr will behave as if istr is at end of stream (that is, will return "" or NONE or true).

[input istr] reads some elements from istr, returning a vector v of those elements. The vector will be empty (size v=0) if and only if istr is at end of stream or is closed. May block (not return until data are available in the external world).

[inputAll istr] reads and returns the vector v of all bytes remaining in istr up to end of stream.

[inputNoBlock istr] returns SOME(v) if some elements v can be read without blocking; returns SOME("") if it can be determined without blocking that istr is at end of stream; returns NONE otherwise. If istr does not support non-blocking input, raises Io.NonblockingNotSupported.

[input1 istr] returns SOME(e) if at least one element e of istr is

BINIO 13

available; returns NONE if istr is at end of stream or is closed; blocks if necessary until one of these conditions holds.

[inputN(istr, n)] returns the next n bytes from istr as a vector, if that many are available; returns all remaining bytes if end of stream is reached before n bytes are available; blocks if necessary until one of these conditions holds.

[endOfStream istr] returns false if any elements are available in istr; returns true if istr is at end of stream or closed; blocks if necessary until one of these conditions holds.

[lookahead istr] returns SOME(e) where e is the next element in the stream; returns NONE if istr is at end of stream or is closed; blocks if necessary until one of these conditions holds. Does not advance the stream.

BYTE OUTPUT:

[openOut s] creates a new outstream associated with the file named s. If file s does not exist, and the directory exists and is writable, then a new file is created. If file s exists, it is truncated (any existing contents are lost).

[openAppend s] creates a new outstream associated with the file named s. If file s does not exist, and the directory exists and is writable, then a new file is created. If file s exists, any existing contents are retained, and output goes at the end of the file.

[closeOut ostr] closes stream ostr; further operations on ostr (except for additional close operations) will raise exception Io.Io.

[output(ostr, v)] writes the byte vector v on outstream ostr.

[output1(ostr, e)] writes the byte e on outstream ostr.

[flushOut ostr] flushes the outstream ostr, so that all data written to ostr becomes available to the underlying file or device.

The functions below are not yet implemented:

 $[\mathtt{setPosIn}(\mathtt{istr},\ \mathtt{i})]$ sets istr to the position i. Raises Io.Io if not supported on istr.

[getPosIn istr] returns the current position of istr. Raises Io.Io if not supported on istr.

[endPosIn istr] returns the last position of istr.

[getPosOut ostr] returns the current position in stream ostr. Raises Io.Io if not supported on ostr.

[endPosOut ostr] returns the ending position in stream ostr. Raises Io.Io if not supported on ostr.

[setPosOut(ostr, i)] sets the current position in stream to ostr to i. Raises Io.Io if not supported on ostr.

 $[mkInstream\ sistr]$ creates a state-based instream from the functional instream sistr.

[getInstream istr] returns the functional instream underlying the state-based instream istr.

[setInstream(istr, sistr)] redirects istr, so that subsequent input is taken from the functional instream sistr.

[mkOutstream sostr] creates a state-based outstream from the outstream sostr.

14 BINIO

 $[\mbox{\tt getOutstream ostr}]$ returns the outstream underlying the state-based outstream ostr.

[setOutstream(ostr, sostr)] redirects the outstream ostr so that subsequent output goes to sostr.

BINARYMAP 15

Module Binarymap

Binarymap -- applicative maps as balanced ordered binary trees From SML/NJ lib 0.2, copyright 1993 by AT&T Bell Laboratories Original implementation due to Stephen Adams, Southampton, UK

[('key, 'a) dict] is the type of applicative maps from domain type 'key to range type 'a, or equivalently, applicative dictionaries with keys of type 'key and values of type 'a. They are implemented as ordered balanced binary trees.

[mkDict ordr] returns a new, empty map whose keys have ordering ordr.

[insert(m, i, v)] extends (or modifies) map m to map i to v.

[find(m, k)] returns v if m maps k to v; otherwise raises NotFound.

[peek(m, k)] returns SOME v if m maps k to v; otherwise returns NONE.

[remove(m, k)] removes k from the domain of m and returns the modified map and the element v corresponding to k. Raises NotFound if k is not in the domain of m.

[numItems m] returns the number of entries in m (that is, the size of the domain of m).

[listItems m] returns a list of the entries $(k,\ v)$ of keys k and the corresponding values v in m, in order of increasing key values.

[app f m] applies function f to the entries $(k,\ v)$ in m, in increasing order of k (according to the ordering ordr used to create the map or dictionary).

[revapp f m] applies function f to the entries $(k,\ v)$ in m, in decreasing order of k.

[foldl f e m] applies the folding function f to the entries $(k,\ v)$ in m, in increasing order of k.

[foldr f e m] applies the folding function f to the entries $(k,\ v)$ in m, in decreasing order of k.

[map f m] returns a new map whose entries have form (k, f(k,v)), where (k, v) is an entry in m.

[transform f m] returns a new map whose entries have form $(k,\ f\ v)$, where $(k,\ v)$ is an entry in m.

16 BINARYSET

Module Binaryset

Binaryset -- sets implemented by ordered balanced binary trees From SML/NJ lib 0.2, copyright 1993 by AT&T Bell Laboratories Original implementation due to Stephen Adams, Southampton, UK

type 'item set exception NotFound val empty : ('item * 'item -> order) -> 'item set : ('item * 'item -> order) -> 'item -> 'item set val singleton val add : 'item set * 'item -> 'item set : 'item set * 'item list -> 'item set val addList : 'item set * 'item -> 'item val retrieve : 'item set * 'item -> 'item option val peek
val isEmpty : 'item set -> bool : 'item set * 'item set -> bool val equal : 'item set * 'item set -> bool val isSubset : 'item set * 'item -> bool val member : 'item set * 'item -> 'item set val delete : 'item set -> int val numItems : 'item set * 'item set -> 'item set val union val intersection : 'item set * 'item set -> 'item set val difference : 'item set * 'item set -> 'item set : 'item set -> 'item list val listItems : ('item -> unit) -> 'item set -> unit val app : ('item -> unit) -> 'item set -> unit val revapp : ('item * 'b -> 'b) -> 'b -> 'item set -> 'b
: ('item * 'b -> 'b) -> 'b -> 'item set -> 'b val foldr val foldl : ('item -> bool) -> 'item set -> 'item option val find ['item set] is the type of sets of ordered elements of type 'item. The ordering relation on the elements is used in the representation of the set. The result of combining two sets with different underlying ordering relations is undefined. The implementation uses ordered balanced binary trees. [empty ordr] creates a new empty set with the given ordering relation. [singleton ordr i] creates the singleton set containing i, with the given ordering relation. [add(s, i)] adds item i to set s. [addList(s, xs)] adds all items from the list xs to the set s. [retrieve(s, i)] returns i if it is in s; raises NotFound otherwise. [peek(s, i)] returns SOME i if i is in s; returns NONE otherwise. [isEmpty s] returns true if and only if the set is empty. [equal(s1, s2)] returns true if and only if the two sets have the same elements. [isSubset(s1, s2)] returns true if and only if s1 is a subset of s2. [member(s, i)] returns true if and only if i is in s. [delete(s, i)] removes item i from s. Raises NotFound if i is not in s. [numItems s] returns the number of items in set s. [union(s1, s2)] returns the union of s1 and s2. [intersection(s1, s2)] returns the intersection of s1 and s2. [difference(s1, s2)] returns the difference between s1 and s2 (that is, the set of elements in s1 but not in s2).

BINARYSET 17

[listItems s] returns a list of the items in set s, in increasing order.

[app f s] applies function f to the elements of s, in increasing order.

[revapp f s] applies function f to the elements of s, in decreasing order.

[foldl f e s] applies the folding function f to the entries of the set in increasing order.

[foldr f e s] applies the folding function f to the entries of the set in decreasing order.

[find p s] returns SOME i, where i is an item in s which satisfies p, if one exists; otherwise returns NONE.

18 BOOL

Module Bool

Bool -- SML Basis Library

datatype bool = datatype bool

val not : bool -> bool

val toString : bool -> string

val fromString : string -> bool option
val scan : (char, 'a) StringCvt.reader -> (bool, 'a) StringCvt.reader

[bool] is the type of Boolean (logical) values: true and false.

[not b] is the logical negation of b.

[toString b] returns the string "false" or "true" according as b is false or true.

[fromString s] scans a boolean b from the string s, after possible initial whitespace (blanks, tabs, newlines). Returns (SOME b) if s has a prefix which is either "false" or "true"; the value b is the corresponding truth value; otherwise NONE is returned.

[scan getc src] scans a boolean b from the stream src, using the stream accessor getc. In case of success, returns SOME(b, rst) where b is the scanned boolean value and rst is the remainder of the stream; otherwise returns NONE.

BUFFER 19

Module Buffer

```
signature Buffer =
sig

   type buf
   val new    : int -> buf
   val contents : buf -> string
   val size    : buf -> int
   val clear    : buf -> unit
   val reset    : buf -> unit

   val addChar          : buf -> char -> unit
   val addString          : buf -> string -> unit
   val addSubString : buf -> substring -> unit
end
```

concatenation at the end and automatically expand as necessary. It provides accumulative concatenation of strings in quasi-linear time (instead of quadratic time when strings are concatenated pairwise).

[new hint] creates a new empty buffer. Raises Size if hint <= 0 or hint > String.maxSize.

The argument hint is used as the initial size of the internal string that holds the buffer contents. The internal string is automatically reallocated as sontents is stored in the buffer. For best performance, hint should be of the same order of magnitude as the number of characters that are expected to be stored in the buffer (for instance, 80 for a buffer that holds one output line). Nothing bad will happen if the buffer grows beyond that limit, however. In doubt, take hint = 16 for instance.

[contents buf] returns the contents of buf.

[size buf] returns the size of the contents of buf.

[clear buf] emptys buf.

[reset buf] emptys buf and shrink the internal string to the initial hint.

[addChar buf c] appends c at the end of buf.

[addString buf s] appends s at the end of buf.

[addSubString buf ss] appends ss at the end of buf.

20 BYTE

Module Byte

Byte -- SML Basis Library

val unpackStringVec : Word8Vector.vector * int * int option -> string
val unpackString : Word8Array.array * int * int option -> string
val packString : Word8Array.array * int * Substring.substring -> unit

Conversions between bytes and characters, and between byte vectors and strings (character vectors).

[byteToChar w] is the character corresponding to the byte w.

[charToByte c] is the byte corresponding to character c.

[bytesToString v] is the string whose character codes are the bytes from vector v.

[stringToBytes s] is the byte vector of character codes of the string s.

In Moscow ${\rm ML}$, all the above operations take constant time. That is, no copying is done.

[unpackStringVec (v, i, NONE)] is the string whose character codes are the bytes of v[i..length v-1]. Raises Subscript if i<0 or i>length v.

[unpackStringVec (v, i, SOME n)] is the string whose character codes are the bytes of v[i..i+n-1]. Raises Subscript if i<0 or n<0 or i+n>length v.

[unpackString (a, i, NONE)] is the string whose character codes are the bytes of a[i..length a-1]. Raises Subscript if i<0 or i>length a.

[unpackString (a, i, SOME n)] is the string whose character codes are the bytes of a[i..i+n-1]. Raises Subscript if i<0 or n<0 or i+n>length a.

[packString (a, i, ss)] copies the character codes of substring ss into the subarray a[i..i+n-1] where n = Substring.size ss. Raises Subscript if i<0 or i+n > length a.

CALLBACK 21

Module Callback

Callback -- registering ML values with C, and accessing C values from ML Registering ML values for access from C code: : string -> 'a -> unit val register val unregister : string -> unit val isRegistered : string -> bool Accessing C variables and functions from ML: type cptr val getcptr : string -> cptr : cptr -> 'b val var : cptr -> 'a1 -> 'b val app1 : cptr -> 'a1 -> 'a2 -> 'b : cptr -> 'a1 -> 'a2 -> 'a3 -> 'b : cptr -> 'a1 -> 'a2 -> 'a3 -> 'a4 -> 'b val app2 val app3 val app4 : cptr -> 'a1 -> 'a2 -> 'a3 -> 'a4 -> 'a5 -> 'b val app5 REGISTERING ML VALUES FOR ACCESS FROM C CODE This example shows how to register the ML function (fn n => 2*n) so that it may be called from C code. (0) The ML side registers the function: Callback.register "myfun" (fn n => 2*n) (1) The C side first obtains an ML value pointer: valueptr mvp = get valueptr("myfun"); (2) The C side then uses the ML value pointer to obtain an ML value, and uses it: callback(get_value(mvp), Val_long(42)); Operation (1) involves a callback to ML, and hence may be slow. Calling get_valueptr may cause the garbage collector to run; hence other live ML values must be registered as GC roots. The garbage collector will never move the ML value pointer; hence it need not be registered as a GC root in the C code. Operation (2) is very fast. If the garbage collector is invoked between the call of $get_value()$ and the use of the ML value, then the value must be registered as a GC root. However, the idiom callback(get_value(mvp), arg1); is safe provided the evaluation of arg1 does not provoke a garbage collection (e.g. if argl is a variable). The C function get_valueptr returns NULL if nam is not registered. The C function get_value returns NULL if nam has been unregistered (and not reregistered) since mvp was obtained; it raises exception Fail if mvp itself is NULL. Every access to the ML value from C code should use the ML valueptr and get valueptr, otherwise the C code will not know when the value has been unregistered and possibly deallocated. The C functions (in mosml/src/runtime/callback.c) void registervalue(char* nam, value mlval) void unregistervalue(char* nam) can be used just as Callback.register and Callback.unregister. The C functions value callbackptr (valueptr mvp, value arg1) value callbackptr2(valueptr mvp, value arg1, value arg2)
value callbackptr3(valueptr mvp, value arg1, value arg2, value arg3)
can be used for callback via an ML value pointer; they will raise

22 CALLBACK

exception Fail if the ML function indicated by mvp has been unregistered.

[register nam v] registers the ML value v, so that it can be retrieved from C code under the name nam. If nam has previously been registered and then unregistered, it will be reregistered with the new value. The new value immediately becomes visible to the C side, both via get_valueptr(nam) and via any ML value pointer previously obtained for nam. Raises exception Fail if nam has been registered and not yet unregistered.

[unregister nam] deletes the registration. This prevents C code from obtaining an ML value pointer for nam and from using an ML value pointer already obtained (but does not prevent C from attempting to use a stored ML value previously obtained with the help of the ML value pointer, which is unsafe anyway). Does nothing if nam is already unregistered. Raises exception Fail if nam has never been registered.

[isRegistered nam] returns true if nam has been registered and not yet unregistered.

ACCESSING REGISTERED C VARIABLES AND FUNCTIONS FROM ML

This example shows how to register the C function

```
value silly_cfun(value v)
{ return copy_double(42.42 * Double_val(v)); }
```

so that it may be called from ML.

- (0) The C side registers the function: registercptr("mycfun", sillycfun);
- (1) The ML side obtains a C pointer and defines an ML function via that pointer: val sillycfun = appl (getcptr "mycfun") : real -> real The type ascription is needed to ensure any type safety whatsoever. Mistakes in the types will lead to crashes, as usual with C.
- (2) To the ML side, the new ML function is indistinguishable from other ML functions val result = sillyfun(3.4)

The C function (in mosml/src/runtime/callback.c)
 void registercptr(char* nam, void* cptr);

is used to register C pointers for access from ML. Only pointers to static C variables, and C functions, should be registered. There is no way to unregister a C pointer.

[cptr] is the type of pointers to C variables and C functions.

[getcptr nam] returns a pointer to the C variable or function registered (by the C side) under the name nam. Raises exception Fail if the name nam has not been registered.

[var cptr] returns the value of the C variable associated with cptr.

[appl cptr arg1] applies the C function associated with cptr to arg1.

[app2 cptr arg1 arg2] applies the C function associated with cptr to (arg1, arg2).

[app3 cptr arg1 arg2 arg3] applies the C function associated with cptr to (arg1, arg2, arg3).

[app4 cptr arg1 arg2 arg3 arg4] applies the C function associated with cptr to (arg1, arg2, arg3, arg4).

CALLBACK 23

[app5 cptr arg1 arg2 arg3 arg4 arg5] applies the C function associated with cptr to (arg1, arg2, arg3, arg4, arg5).

24 CHAR

Module Char

type char = char

Char -- SML Basis Library

```
val minChar : char
val maxChar : char
val maxOrd : int
val chr
           : int -> char
                                 May raise Chr
val ord
            : char -> int
val succ
            : char -> char
                                 May raise Chr
            : char -> char
                                 May raise Chr
val pred
                : char -> bool
: char -> bool
                                 contains "abcdefghijklmnopqrstuvwxyz"
contains "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
val isLower
val isUpper
                                  contains "0123456789"
                : char -> bool
val isDigit
                : char -> bool
                                  isUpper orelse isLower
val isAlpha
val isHexDigit
                : char -> bool
                                  isDigit orelse contains "abcdefABCDEF"
val isAlphaNum : char -> bool
                                  isAlpha orelse isDigit
                                  any printable character (incl. #" ")
val isPrint
                : char -> bool
val isSpace
val isPunct
                                 contains " \t \n \" printable, not space or alphanumeric
                : char -> bool
                : char -> bool
                : char -> bool
val isGraph
                                  (not isSpace) andalso isPrint
                : char -> bool
: char -> bool
                                  ord\ c < 128
val isAscii
val isCntrl
                                  control character
val toLower
                : char -> char
val toUpper
                : char -> char
val fromString : string -> char option
                                               ML escape sequences
                : char -> string
val toString
                                               ML escape sequences
val fromCString : string -> char option
                                               C escape sequences
val toCString : char -> string
                                               C escape sequences
              : string -> char -> bool
val contains
val notContains : string -> char -> bool
            : char * char -> bool
val <
            : char * char -> bool
val <=
            : char * char -> bool
val >
           : char * char -> bool
val >=
val compare : char * char -> order
   [char] is the type of characters.
   [minChar] is the least character in the ordering <.
   [maxChar] is the greatest character in the ordering <.
   [maxOrd] is the greatest character code; equals ord(maxChar).
   [chr i] returns the character whose code is i. Raises Chr if
   i<0 or i>maxOrd.
   [ord c] returns the code of character c.
   [succ c] returns the character immediately following c, or raises
   Chr if c = maxChar.
   [pred c] returns the character immediately preceding c, or raises
   Chr if c = minChar.
   [isLower c] returns true if c is a lowercase letter (a to z).
   [isUpper c] returns true if c is a uppercase letter (A to Z).
   [isDigit c] returns true if c is a decimal digit (0 to 9).
```

CHAR 25

[isAlpha c] returns true if c is a letter (lowercase or uppercase).

[isHexDigit c] returns true if c is a hexadecimal digit (0 to 9 or a to f or A to F).

[isAlphaNum c] returns true if c is alphanumeric (a letter or a decimal digit).

[isPrint c] returns true if c is a printable character (space or visible)

[isSpace c] returns true if c is a whitespace character (blank, newline, tab, vertical tab, new page).

[isGraph c] returns true if c is a graphical character, that is, it is printable and not a whitespace character.

[isPunct c] returns true if c is a punctuation character, that is, graphical but not alphanumeric.

[isCntrl c] returns true if c is a control character, that is, if not (isPrint c).

[isAscii c] returns true if 0 <= ord c <= 127.

[toLower c] returns the lowercase letter corresponding to c, if c is a letter (a to z or A to Z); otherwise returns c.

[toUpper c] returns the uppercase letter corresponding to c, if c is a letter (a to z or A to Z); otherwise returns c.

[contains s c] returns true if character c occurs in the string s; false otherwise. The function, when applied to s, builds a table and returns a function which uses table lookup to decide whether a given character is in the string or not. Hence it is relatively expensive to compute val p = contains s but very fast to compute p(c) for any given character.

[notContains s c] returns true if character c does not occur in the string s; false otherwise. Works by construction of a lookup table in the same way as the above function.

[fromString s] attempts to scan a character or ML escape sequence from the string s. Does not skip leading whitespace. For instance, fromString " $\065$ " equals #"A".

[toString c] returns a string consisting of the character c, if c is printable, else an ML escape sequence corresponding to c. A printable character is mapped to a one-character string; bell, backspace, tab, newline, vertical tab, form feed, and carriage return are mapped to the two-character strings "\\a", "\\b", "\\t", "\\n", "\\v", "\f", and "\\r"; other characters with code less than 32 are mapped to three-character strings of the form "\\^Z", and characters with codes 127 through 255 are mapped to four-character strings of the form "\\ddd", where ddd are three decimal digits representing the character code. For instance,

toString #"A" equals "A" toString #"\\" equals "\\\\" toString #"\"" equals "\\\"" 0) equals "\\^@" toString (chr toString (chr 1) equals "\\^A" 6) equals "\\^F" toString (chr toString (chr 7) equals "\\a" 8) equals "\\b" toString (chr 9) equals "\\t" toString (chr toString (chr 10) equals "\\n" 11) equals "\\v" toString (chr toString (chr 12) equals " \f " 13) equals "\\r" toString (chr toString (chr 14) equals "\\^N" toString (chr 127) equals "\\127" toString (chr 128) equals "\\128" 26 CHAR

[fromCString s] attempts to scan a character or C escape sequence from the string s. Does not skip leading whitespace. For instance, fromString " $\065$ " equals #"A".

[toCString c] returns a string consisting of the character c, if c is printable, else an C escape sequence corresponding to c. A printable character is mapped to a one-character string; bell, backspace, tab, newline, vertical tab, form feed, and carriage return are mapped to the two-character strings "\\a", "\\b", "\\t", "\\r", "\\r", and "\\r"; other characters are mapped to four-character strings of the form "\\ooo", where ooo are three octal digits representing the character code. For instance,

```
toString #"A"
                     equals "A"
toString #"A"
                     equals "A"
toString #"\\"
toString #"\"
                     equals "\\\"
equals "\\\"
toString (chr 0) equals "\\000"
toString (chr 1) equals "\\001"
                 6) equals "\\006"
7) equals "\\a"
toString (chr
toString (chr
                8) equals "\\b"
toString (chr
                 9) equals "\\t"
toString (chr
                 10) equals "\\n"
toString (chr
toString (chr 11) equals "\\v"
toString (chr 12) equals "\\f"
toString (chr 13) equals "\\r" toString (chr 14) equals "\\016"
toString (chr 127) equals "\\177"
toString (chr 128) equals "\\200"
```

[<] [<=] [>]

[>=] compares character codes. For instance, c1 < c2 returns true if ord(c1) < ord(c2), and similarly for <=, >, >=.

[compare(c1, c2)] returns LESS, EQUAL, or GREATER, according as c1 is precedes, equals, or follows c2 in the ordering Char. < .

CHARARRAY 27

Module CharArray

```
CharArray -- SML Basis Library
eqtype array
type elem = Char.char
type vector = CharVector.vector
val maxLen : int
           : int * elem -> array
val array
val tabulate : int * (int -> elem) -> array
val fromList : elem list -> array
             : array -> int
val length
             : array * int -> elem
: array * int * elem -> unit
val sub
val update
val vector
             : array -> vector
val copy : {src: array, dst: array, di: int} -> unit val copyVec : {src: vector, dst: array, di: int} -> unit
: (elem -> bool) -> array -> bool
val all
             : (elem -> unit) -> array -> unit
: (elem * 'b -> 'b) -> 'b -> array -> 'b
: (elem * 'b -> 'b) -> 'b -> array -> 'b
: (elem -> elem) -> array -> unit
val foldl
val foldr
val modify
val findi
             : (int * elem -> bool) -> array -> (int * elem) option
val modifyi : (int * elem -> elem) -> array -> unit
val collate : (elem * elem -> order) -> array * array -> order
```

[array] is the type of one-dimensional, mutable, zero-based constant-time-access arrays with elements of type Char.char, that is, characters. Arrays al and a2 are equal if both were created by the same call to a primitive, or if both are empty.

All operations are as for Array.array.

28 CHARARRAYSLICE

Module CharArraySlice

```
CharArraySlice -- SML Basis Library
type elem = char
type array = CharArray.array
type vector = CharVector.vector
type vector_slice = CharVectorSlice.slice
type slice
              : slice -> int
val length
val sub
              : slice * int -> elem
              : slice * int * elem -> unit
val update
              : array * int * int option -> slice
val slice
val full : array -> slice
val subslice : slice * int * int option -> slice
val base
              : slice -> array * int * int
              : slice -> vector
val vector
              : {src: slice, dst: array, di: int} -> unit
val copy
             : {src: vector_slice, dst: array, di: int} -> unit
val copyVec
val isEmpty : slice -> bool
val getItem : slice -> (elem * slice) option
val find
              : (elem -> bool) -> slice -> elem option
              : (elem -> bool) -> slice -> bool
: (elem -> bool) -> slice -> bool
val exists
val all
             : (elem -> unit) -> slice -> unit
: (elem * 'b -> 'b) -> 'b -> slice -> 'b
: (elem * 'b -> 'b) -> 'b -> slice -> 'b
val app
val foldl
val foldr
             : (elem -> elem) -> slice -> unit
val modify
              : (int * elem -> bool) -> slice -> (int * elem) option
val findi
              : (int * elem -> unit) -> slice -> unit
val appi
              : (int * elem * 'b -> 'b) -> 'b -> slice -> 'b
val foldli
              : (int * elem * 'b -> 'b) -> 'b -> slice -> 'b
val foldri
val modifyi : (int * elem -> elem) -> slice -> unit
val collate : (elem * elem -> order) -> slice * slice -> order
```

All operations are as for ArraySlice.slice.

CHARVECTOR 29

Module CharVector

```
CharVector -- SML Basis Library
type vector = string
type elem = Char.char
val maxLen : int
val fromList : elem list -> vector
val tabulate : int * (int -> elem) -> vector
val length
            : vector -> int
            : vector * int -> elem
val sub
           : vector * int * elem -> vector
val update
            : vector list -> vector
val concat
val find
            : (elem -> bool) -> vector -> elem option
val app
            : (elem -> unit) -> vector -> unit
           : (elem -> elem) -> vector -> vector
: (elem * 'b -> 'b) -> 'b -> vector -> 'b
: (elem * 'b -> 'b) -> 'b -> vector -> 'b
val map
val foldl
val foldr
            : (int * elem -> bool) -> vector -> (int * elem) option
val findi
val appi
           : (int * elem -> unit) -> vector -> unit
val collate : (elem * elem -> order) -> vector * vector -> order
```

[vector] is the type of one-dimensional, immutable, zero-based constant-time-access vectors with elements of type Char.char, that is, characters. Type vector admits equality, and vectors v1 and v2 are equal if they have the same length and their elements are equal. The type vector is the same as String.string.

All operations are as for Vector.vector.

30 CHARVECTORSLICE

Module CharVectorSlice

```
CharVectorSlice -- SML Basis Library
type elem = Char.char
type vector = CharVector.vector
type slice = Substring.substring
              : slice -> int
val length
               : slice * int -> elem
val sub
               : vector * int * int option -> slice
val slice
val full : vector -> slice
val subslice : slice * int * int option -> slice
              : slice -> vector * int * int
val base
val vector
               : slice -> vector
               : slice list -> vector
val concat
val isEmpty : slice -> bool
val getItem : slice -> (elem * slice) option
val find
              : (elem -> bool) -> slice -> elem option
val exists : (elem -> bool) -> slice -> bool
               : (elem -> bool) -> slice -> bool
val all
val app
               : (elem -> unit) -> slice -> unit
               : (elem -> elem) -> slice -> vector
: (elem * 'b -> 'b) -> 'b -> slice -> 'b
val map
val foldl
              : (elem * 'b -> 'b) -> 'b -> slice -> 'b
val foldr
               : (int * elem -> bool) -> slice -> (int * elem) option
val findi
               : (int * elem -> unit) -> slice -> unit
val appi
              : (int * elem -> elem) -> slice -> unit

: (int * elem -> elem) -> slice -> vector

: (int * elem * 'b -> 'b) -> 'b -> slice -> 'b

: (int * elem * 'b -> 'b) -> 'b -> slice -> 'b
val mapi
val foldli
val foldri
val collate : (elem * elem -> order) -> slice * slice -> order
```

[slice] is the type of CharVector slices, that is, sub-vectors of CharVector.vector values. Since a CharVector.vector is a string, a slice is the same as a substring, and slices may be processed using the functions defined as well as those in structure Substring.

The slice (a,i,n) is valid if 0 <= i <= i+n <= size s, or equivalently, 0 <= i and 0 <= n and i+n <= size s. A valid slice sli = (a,i,n) represents the sub-vector a[i...i+n-1], so the elements of sli are a[i], a[i+1], ..., a[i+n-1], and n is the length of the slice. Only valid slices can be constructed by these functions.

All operations are as for VectorSlice.slice.

COMMANDLINE 31

Module CommandLine

CommandLine -- SML Basis Library

[name ()] returns the name used to start the current process.

[arguments ()] returns the command line arguments of the current process. Hence List.nth(arguments (), 0) is the first argument.

32 DATE

Module Date

```
Date -- SML Basis Library
datatype weekday = Mon | Tue | Wed | Thu | Fri | Sat | Sun
datatype month = Jan | Feb | Mar | Apr | May |
                 | Jul | Aug | Sep | Oct | Nov | Dec
type date
exception Date
val date : {
               year
                       : int,
                                               e.g. 1999
               month : month,
                                              Jan, Feb, ...
                      : int,
: int,
                                               1-31
                                               0-23
               hour
               minute : int,
                                               0 - 59
               \begin{array}{lll} \text{second} : & \text{int,} & \textit{0-61, permitting leap seconds} \\ \text{offset} : & \text{Time.time option} & \textit{time zone west of UTC} \end{array}
             } -> date
                 : date -> int
val year
                 : date -> month
val month
                 : date -> int
val day
                : date -> int
val hour
val minute
                : date -> int
                : date -> int
: date -> weekday
val second
val weekDay
                : date -> int
val yearDay
                : date -> bool option
val isDst
                 : date -> Time.time option
val offset
                 : date * date -> order
val compare
val toString : date -> string
                 : string -> date -> string
val fmt
val fromString : string -> date option
                 : (char, 'a) StringCvt.reader -> (date, 'a) StringCvt.reader
val fromTimeLocal : Time.time -> date
val fromTimeUniv : Time.time -> date
val toTime
val localOffset : date -> Time.time
val localOffset : unit -> Time.time
```

These functions convert times to dates and vice versa, and format and scan dates.

[date] is the type of points in time in a given time zone. If the offset is NONE, then the date is in the local time zone. If the offset is SOME t, then t is the offset of the main timezone (ignoring daylight savings time) west of UTC.

When 0 hours <= t < 12 hours, the represented time is to the west of UTC and the local time is UTC-t.

When 12 hours <= t < 23 hours, the represented time is to the East of UTC and the local time is UTC+(24-t).

[date { year, month, day, hour, minute, second, offset }] returns a canonical date value. Seconds outside the range 0..59 are converted to the equivalent minutes and added to the minutes argument; leap seconds are ignored. Similarly, excess minutes are converted to hours, hours to days, days to months, and months to years. Then the weekday and day number in the year are computed. Leap years are assumed in accordance with the Gregorian calendar, for any year after year 0 A.D.

If the offset is greater than one day (24 hours), then the excess days are added to the days, and the offset modulo 24 hours is used.

[year dt] returns the year of dt, e.g. 1999. [month dt] returns the month of dt. [day dt] returns the day of dt [hour dt] returns the hour of dt. [minute dt] returns the minute of dt. [second dt] returns the second of dt. [weekDay dt] returns the weekday of dt. [yearDay dt] returns the number of the day in the year of dt. January 1 is day 0, and December 31 is day 364 (and 365 in leap years). [isDst dt] returns SOME(true) if daylight savings time is in effect at the date dt; returns SOME(false) if not; and returns NONE if this information is unavailable. [offset dt] returns NONE if the date dt is in the local time zone; returns SOME t where t is the offset west of UTC otherwise. SOME(Time.zeroTime) is UTC. [compare(dt1, dt2)] returns LESS, EQUAL, or GREATER, according as date dt1 precedes, equals, or follows dt2 in time. Lexicographically compares the dates. Ignores timezone offset and DST. Does not detect invalid dates. [toString dt] returns a 24 character string representing the date dt in the following format: Wed Mar 8 19:06:45 1995 The result may be wrong if the date is not representable as a Time.time value. Raises Date if dt is an invalid date. Corresponds to the ANSI C function 'asctime'. [fmt fmtstr dt] formats the date dt according to the format string fmtstr. The format string has the same meaning as with the ANSI ${\tt C}$ function 'strftime'. These ANSI C format codes should work on all platforms: %a abbreviated weekday name (e.g. "Mon")
%A full weekday name (e.g. "Monday") %b abbreviated month name (e.g. "Oct") %B full month name (e.g. "October")
%c date and time (e.g. "Dec 2 06:55:15 1979") %d day of month (01..31) %H hour (00..23) %I hour (01..12) %j day of year (001..366) %m month number (01..12) %M minutes (00..59) locale's equivalent of a.m./p.m. S seconds (00..61, allowing for leap seconds) %U week number (00..53), with Sunday as the first day of week 01 %w day of week, with 0 representing Sunday (0..6)
%W week number (00..53), with Monday as the first day of week 01 %x locale's appropriate date representation %y year of century (00..99) %Υ year including century (e.g. 1997) %Z time zone name if it exists; otherwise the empty string %% the percent character Example: The current local date in ISO format (e.g. 1998-04-06) can be obtained by using: fmt "%Y-%m-%d" (fromTimeLocal (Time.now ())) [fromString s] scans a 24-character date from the string s, after possible initial whitespace (blanks, tabs, newlines). The format of the string must be as produced by toString. The fields isDst and offset in the resulting date will be NONE. No check of the

34 DATE

consistency of the date (weekday, date in the month, \dots) is performed.

[scan getc src] scans a 24-character date from the stream src, using the stream accessor getc. Otherwise works as fromString. In case of success, returns SOME(date, rst) where date is the scanned date and rst is the remainder of the stream; otherwise returns NONE

[fromTimeLocal t] returns the local date at (UTC) time t. The resulting date will have offset = NONE. The fields year, month, day, hour, minute, and second are as expected. The resulting isDst may be NONE if the system cannot determine whether daylight savings time is in effect at the given time. Corresponds to the ANSI C function 'localtime'.

[fromTimeUniv t] is similar to fromTime, but returns the UTC date at (UTC) time t. The resulting date will have offset = SOME Time.zeroTime. Corresponds to the ANSI C function 'gmtime'.

[toTime dt] returns the (UTC) time corresponding to the date dt. Uses the isDst time field if it is present (SOME _) and cannot be calculated from the given date. May raise Date if the given date is invalid. Raises Time.Time if the Date cannot be represented as a Time.time value. At least the dates in the interval 1970-2030 can be represented as Time.time values. Corresponds to the ANSI C function 'mktime'.

[localOffset ()] is the local time zone offset west of UTC. It holds that 0 hours <= localOffset () < 24 hours.

DYNARRAY 35

Module Dynarray

Dynarray -- polymorphic dynamic arrays a la SML/NJ library

val array
val array : int * '_a -> '_a array
val subArray : '_a array * int * int -> '_a array
val fromList : '_a list * '_a -> '_a array
val tabulate : int * (int -> '_a) * '_a -> '_a array
val sub : 'a array * int -> 'a
val update : '_a array * int * '_a -> unit
val default : 'a array -> 'a
val bound : 'a array -> int

['ty array] is the type of one-dimensional, mutable, zero-based unbounded arrays with elements of type 'ty. Type 'ty array does not admit equality.

[array(n, d)] returns a dynamic array, all of whose elements are initialized to the default d. The parameter n is used as a hint of the upper bound on non-default elements. Raises Size if n < 0.

[subArray(a, m, n)] returns a new array with the same default value as a, and whose values in the range [0,n-m] equal the values in a in the range [m,n]. Raises the exception Size if n < m.

[fromList (xs, d)] returns an array whose first elements are those of [xs], and the rest are the default d.

[tabulate(n, f, d)] returns a new array whose first n elements are f 0, f 1, ..., f (n-1), created from left to right, and whose remaining elements are the default d. Raises Size if n < 0.

[sub(a, i)] returns the i'th element of a, counting from 0. Raises Subscript if i < 0.

[update(a, i, x)] destructively replaces the i'th element of a by x. Raises Subscript if i < 0.

[default a] returns the default value of the array a.

[bound a] returns an upper bound on the indices of non-default values.

36 DYNLIB

Module Dynlib

```
Dynlib -- dynamic linking with foreign functions
type dlHandle
type symHandle
exception Closed
datatype flag = RTLD_LAZY | RTLD_NOW
val dlopen : { lib : string, flag : flag, global : bool } -> dlHandle
val dlsym : dlHandle -> string -> symHandle
val dlclose : dlHandle -> unit
val var : symHandle -> 'b
val app1 : symHandle -> 'a1 -> 'b
val app2 : symHandle -> 'a1 -> 'a2 -> 'b
val app3 : symHandle -> 'a1 -> 'a2 -> 'a3 -> 'b
val app4 : symHandle -> 'a1 -> 'a2 -> 'a3 -> 'a4 -> 'b
val app5 : symHandle -> 'a1 -> 'a2 -> 'a3 -> 'a4 -> 'a5 -> 'b
   Structure Dynlib provides dynamic loading and calling of C
   functions, using the dlfcn interface. A dynamic library is a
   collection of symbols (C variables and functions).
   An ML value passed to or returned from a symbol has type 'value' as
  defined in src/runtime/mlvalues.h. The C functions should use the
   macroes defined there to access and produce ML values. When
   writing a C function, remember that the garbage collector may be
  activated whenever you allocate an ML value. Also, remember that
   the garbage collector may move values from the young heap to the
   old one, so that a C pointer pointing into the ML heap may need to
  be updated. Use the Push_roots and Pop_roots macroes to achieve
   [dlHandle] is the type of dynamic library handles. A dynamic
   library handle is created by opening a dynamic library using
   dlopen. This will load the library into the runtime system.
   dynamic library handle is used for accessing symbols in that
   library. The library may be closed and removed from the runtime
   system using dlclose.
  The same library may be opened more than once, resulting in
  different library handles. The physical library will be loaded
   only once, though, and will remain in the runtime system until all
   handles to the library have been closed.
   [symHandle] is the type of symbol handles. A symbol handle is used
   to access a symbol (variable or function) in the dynamic library,
   using the functions var, app1, app2, ..., app5. Type safety is the
   responsibility of the programmer; the runtime system performs no
   type checking. Hence you are advised to add explicit types
   whenever you define an ML function in terms of var, app1, ..., app5.
  How to create a dynamically loadable library
  Assume file "xyz.c" contains your C functions.
   To compile xyz.c into xyz.o and then create a dynamic library
   libxyz.so from xyz.o:
     Under Linux and OSF/1 (Digital Unix):
        gcc -c -o xyz.o xyz.c
        ld -shared -o libxyz.so xyz.o
     Under Solaris (ignore the warnings from ld):
        gcc -c -o xyz.o xyz.c
        ld -G -B symbolic -z nodefs -o libxyz.so xyz.o
     Under HP-UX:
        gcc -fPIC -c -o xyz.o xyz.c
        ld -b -B symbolic -E -o libxyz.so xyz.o
```

DYNLIB 37

If "xyz.o" depends on another library "libabc.a" you may link the required functions into libxyz.so just by adding -labc or libabc.a to the above linker command.

If "xyz.o" depends on another dynamic library "libabc.so" you may specify this by adding -labc to the above linker command. Then Dynlib.dlopen will automatically load libabc.so before libxyz.so.

[dlopen { lib, flag, global }] will load and open the library in file 'lib', returning a handle to it. Libraries are usually specified just by file name, leaving out the directory path. Linux/Unix-specific information: Libraries are searched for in those directories mentioned in LD_LIBRARY_PATH, those mentioned in /etc/ld.so.cache, in /usr/lib and /lib. (Note that /etc/ld.so.cache is created from /etc/ld.so.conf by running ldconfig; you must be superuser to do that).

If 'global' is true, then the library's global symbols are made available for other libraries subsequently loaded.

[flag] is the type of library loading modes: RTLD_LAZY and RTLD_NOW.

[RTLD_LAZY] specifies that only symbol relocations will be performed when calling dlopen, whereas function relocations will be performed later when a function is invoked for the first time (if ever). This is the normal situation.

[RTLD_NOW] specifies that all function relocations must be performed immediately, also for functions that will never be called. This checks that all functions are defined, but may waste some time.

[dlsym dlh nam] returns a symbol handle for the symbol called 'nam' in the library associated with dlh. Raises Closed if dlh has been closed.

[dlclose dlh] closes the library handle and deallocates the library if there are no more open handles to this library.

The following functions raise Closed if the associated handle has been closed.

[var sym] returns the value of the C variable associated with sym.

[app1 sym arg1] applies the C function associated with sym to arg1.

[app2 sym arg1 arg2] applies the C function associated with sym to (arg1, arg2).

[app3 sym arg1 arg2 arg3] applies the C function associated with sym to (arg1, arg2, arg3).

[app4 sym arg1 arg2 arg3 arg4] applies the C function associated with sym to (arg1, arg2, arg3, arg4).

[app5 sym arg1 arg2 arg3 arg4 arg5] applies the C function associated with sym to (arg1, arg2, arg3, arg4, arg5).

38 FILESYS

Module FileSys

val compare

OS.FileSys -- SML Basis Library type dirstream : string -> dirstream val openDir : dirstream -> string option val readDir val rewindDir : dirstream -> unit val closeDir : dirstream -> unit val chDir : string -> unit val getDir : unit -> string val mkDir : string -> unit val rmDir : string -> unit val isDir : string -> bool val realPath : string -> string val fullPath : string -> string val isLink : string -> bool val readLink : string -> string : string -> Time.time
: string * Time.time option -> unit val modTime val setTime : string -> unit val remove : {old: string, new: string} -> unit val rename datatype access = A_READ | A_WRITE | A_EXEC val access : string * access list -> bool val fileSize : string -> int val tmpName : unit -> string eqtype file id val fileId : string -> file_id : file_id -> word val hash

: file_id * file_id -> order

These functions operate on the file system. They raise OS.SysErr in case of errors.

[openDir p] opens directory p and returns a directory stream for use by readDir, rewindDir, and closeDir. Subsequent calls to readDir will return the directory entries in some unspecified order.

[readDir dstr] returns SOME(s), consuming an entry s from the directory stream if it is non-empty; returns NONE if it is empty (when all directory entries have been read). Only entries distinct from the parent arc and the current arc (that is, .. and . in Unix, DOS, and Windows; see the Path structure) will be returned.

[rewindDir dstr] resets the directory stream as if it had just been opened.

[closeDir dstr] closes the directory stream. All subsequent operations on the stream will raise OS.SysErr.

[chDir p] changes the current working directory to p. This affects calls to the functions use, load, compile in the interactive system, as well as all functions defined in this library. If p specifies a volume name, then this command also changes the current volume (relevant under DOS, Windows, OS/2, etc.).

[getDir ()] returns the name of the current working directory.

[mkDir p] creates directory p on the file system.

[rmDir p] removes directory p from the file system.

FILESYS 39

[isDir p] tests whether p is a directory.

[fullPath p] returns a canonical form of path p, where all occurrences of the arcs ".", "..", "" have been expanded or removed, and (under Unix) symbolic links have been fully expanded. Raises SysErr if a directory on the path, or the file or directory named, does not exist or is not accessible, or if there is a link loop.

[realPath p] behaves as fullPath(p) if p is absolute. If p is relative and on the same volume as the current working directory, it returns a canonical path relative to the current working directory, where superfluous occurrences of the arcs ".", "..", "" have been removed, and (under Unix) symbolic links have been fully expanded. Raises SysErr if a directory on the path, or the file or directory named, does not exist or is not accessible, or if there is a link loop. Raises Path if p is relative and on a different volume than the current working directory.

[isLink p] returns true if p names a symbolic link. Raises SysErr if the file does not exist or there is an access violation. On operating systems without symbolic links, it returns false, or raises SysErr if the file does not exist or there is an access violation.

[readLink p] returns the contents of the symbolic link p. Raises SysErr if p does not exist or is not a symbolic link, or there is an access violation. On operating systems without symbolic links, it raises SysErr.

[modTime p] returns the modification time of file p.

[setTime (p, tmopt)] sets the modification and access time of file p. If tmopt is SOME t, then the time t is used; otherwise the current time, that is, Time.now(), is used.

[remove p] deletes file p from the file system.

[rename {old, new}] changes the name of file 'old' to 'new'.

[access] is the type of access permissions:

[A_READ] specifies read access.

[A_WRITE] specifies write access.

[A_EXEC] specifies permission to execute the file (or directory).

[access (p, accs)] tests the access permissions of file p, expanding symbolic links as necessary. If the list accs of required access permission is empty, it tests whether p exists. If accs contains A_READ, A_WRITE, or A_EXEC, respectively, it tests whether the user process has read, write, or execute permission for the file.

Under Unix, the access test is done with the 'real' user id and group id (as opposed to the 'effective' user id and group id) of the user process. Hence access("file", [A_READ]) may return false, yet the file may be readable by the process, in case the effective user id or group id has been changed by setuid.

[fileSize p] return the size, in bytes, of the file p. Raises SysErr if p does not exist or its directory is not accessible.

[tmpName ()] returns a file name suitable for creating a fresh temporary file. Note that there is no guarantee that the file name will be unique, since a file of that name may be created between the call to tmpName and a subsequent call to openOut which creates the file. The file name will be absolute, usually of the form /tmp/xxxxxxxx provided by POSIX tmpnam (3).

[file_id] is the type of unique identities of file system objects (including device ids and volume ids, but possibly insensitive to

40 FILESYS

volume changes on removable volumes, such as tapes and diskettes). The set of file ids is equipped with a total linear order.

[fileId p] returns the file_id of the file system object named by path p. It holds that fileId p1 = fileId p2 if and only if p1 and p2 name the same file system object.

[hash fid] returns a hashvalue for fid, suitable for use in a hashtable of file ids (and hence files).

If fid1 = fid2 then hash fid1 = hash fid2.

[compare (fid1, fid2)] returns LESS, EQUAL, or GREATER, according as fid1 precedes, equals, or follows fid2 in the total linear order on file ids. This is suitable for e.g. an ordered binary tree of file ids (and hence files).

GDBM 41

Module Gdbm

```
Gdbm -- GNU gdbm persistent string hashtables -- requires Dynlib
type table
datatype openmode =
   READER
                                         read-only access (nonexclusive)
    WRITER
                                         read/write, table must exist
   WRCREAT
                                         read/write, create if necessary
  NEWDB
                                         read/write, create empty table
type datum = string
exception NotFound
exception AlreadyThere
exception NotWriter
exception Closed
exception GdbmError of string
val withtable : string * openmode -> (table -> 'a) -> 'a
val withtables : (string * openmode) list -> (table list -> 'a) -> 'a
val add
               : table -> datum * datum -> unit
               : table -> datum * datum -> unit
val insert
val find
              : table -> datum -> datum
val peek
val hasKey
               : table -> datum -> datum option
               : table -> datum -> bool
              : table -> datum -> unit
val remove
val listKeys
              : table -> datum list
val numItems
               : table -> int
val listItems : table -> (datum * datum) list
              : (datum * datum -> unit) -> table -> unit
val app
               : (datum * datum -> 'a) -> table -> 'a list
val map
              : (datum * datum * 'a -> 'a) -> 'a -> table -> 'a
val fold
val fastwrite : bool ref
val reorganize : table -> unit
   [table] is the type of an opened table. A value of type table can
   be used only in the argument f to the withtable function. This
   makes sure that the table is closed after use.
   [openmode] is the type of opening modes. Read-only access (READER)
   is non-exclusive; read/write access (WRITER, WRCREAT, NEWDB) is
   exclusive.
   [withtable (nam, mod) f] first opens the table db in file nam with
   mode mod, then applies f to db, then closes db. Makes sure to
   close db even if an exception is raised during the evaluation of
   f(db). Raises GdbmError with an informative message in case the
   table cannot be opened. E.g. the table cannot be opened for
   reading if already opened for writing, and cannot be opened for
   writing if already opened for reading.
   A table is only guaranteed to work properly if created by withtable
   using open modes WRCREAT or NEWDB. If you create a table by creating and then opening an empty file, then numItems, listKeys,
   listItems, etc. will raise an exception.
   [withtables nammod f], where nammod = [(nam1, mod1), ..., (namn, modn)],
   is equivalent to
        withtable (nam1, mod1) (fn db1 =>
            withtable (nam2, mod2) (fn db2 =>
                    f [db1, db2, ...]))
   That is, first opens the databases db1, db2, \dots in that order in files nam1, nam2, \dots with modes mod1, mod2, \dots, then applies f to
   [db1, db2, ...], and finally closes [db1, db2, ...]. Makes sure to
   close all databases even if an exception is raised during the
   opening of db1, db2, ... or during the evaluation of f[db1, db2, ...].
```

42 GDBM

[add db (k,v)] adds the pair (k, v) to db. Raises AlreadyThere if there is a pair $(k, _)$ in db already. Raises NotWriter if db is not opened in write mode.

[insert db (k, v)] adds the pair (k, v) to db, replacing any pair $(k, _)$ at k if present. Raises NotWriter if db is not opened in write mode.

[find db k] returns v if the pair (k, v) is in db; otherwise raises NotFound.

[peek db k] returns SOME v if the pair $(k,\ v)$ is in db; otherwise returns NONE.

[hasKey db k] returns true if there is a pair $(k, _)$ in db; otherwise returns false.

[remove db k] deletes the pair $(k, _)$ from the table if present; otherwise raises NotFound. Raises NotWriter if db is not opened in write mode.

[listKeys db] returns a list of all keys in db in an unspecified order.

[numItems db] is the number of (key, value) pairs in db. Equivalent to length(listKeys db).

[listItems db] returns a list of all (key, value) pairs in db in some order. Equivalent to

List.map (fn key => (key, find(db,key))) (listKeys db)

[app f db] is equivalent to List.app f (listItems db), provided the function f does not change the set of keys in the table. Otherwise the effect is unpredictable.

[map f db] is equivalent to List.map f (listItems db), provided the function f does not change the set of keys in the table. Otherwise the result and effect are unpredictable.

[fold f a db] is equivalent to

List.foldr (fn $((k, v), r) \Rightarrow f(k, v, r)$) a (listItems db) provided the function f does not change the set of keys in the table. Otherwise the result and effect are unpredictable.

[fastwrite] can be set to speed up writes to a table. By default, !fastwrite is false and every write to a table will be followed by file system synchronization. This is safe, but slow if you perform thousands of writes. However, if !fastwrite is true when calling withtable, then writes may not be followed by synchronization, which may speed up writes considerably. In any case, the file system is synchronized before withtable returns.

[reorganize db] has no visible effect, but may be called after a lot of deletions to shrink the size of the table file.

GDIMAGE 43

Module Gdimage

```
Gdimage -- creating PNG images -- requires Dynlib
type image
type color
datatype style =
    ColorS of color
  | TransparentS
datatype mode =
    Color of color
   Transparent
    Brushed of image
    Styled of style vector
    StyledBrushed of bool vector * image
  | Tiled of image
datatype font =
    Tinv
    Small
    MediumBold
    Large
  Giant
type rgb = int * int * int
                                    RGB color components, 0..255
type xy = int * int
                                      points (x, y) and sizes (w, h)
              : xy -> rgb -> image
val image
val fromPng : string -> image
val toPng : image -> string -> unit
val stdoutPng : image -> unit
              : image -> xy
val size
val color
                   : image -> rgb -> color
                   : image -> color -> rgb
val rgb
                   val htmlcolors
                                 green : color, lime : color, maroon : color,
                                 navy : color, olive : color, purple : color,
red : color, silver : color, teal : color,
                                 white : color, yellow : color }
val getTransparent : image -> color option
val setTransparent : image -> color -> unit
val noTransparent : image -> unit
val drawPixel
                : image -> mode -> xy -> unit
                : image -> mode -> xy * xy -> unit
: image -> mode -> xy * xy -> unit
val drawLine
val drawRect
                : image -> mode -> xy * xy -> unit
val fillRect
val drawPolygon : image -> mode -> xy vector -> unit
val fillPolygon : image -> mode -> xy vector -> unit
                : image -> mode -> { c : xy, wh : xy, from : int, to : int }
                   -> unit
                : image -> mode -> xy -> unit
val fill
val fillBorder : image -> mode -> xy -> color -> unit
                 : { src : image, srcxy : xy, srcwh : xy,
val copy
                     dst : image, dstxy : xy } -> unit
val copyResize
                : { src : image, srcxy : xy, srcwh : xy,
                     dst : image, dstxy : xy, dstwh : xy } -> unit
val char
                : image -> color -> font -> xy -> char -> unit
val charUp
                : image -> color -> font -> xy -> char -> unit
                : image -> color -> font -> xy -> string -> unit
val string
val stringUp
                : image -> color -> font -> xy -> string -> unit
                : font -> xy
val charsize
```

44 **GDIMAGE**

This is an interface to version 1.7.3 of Thomas Boutell's gd image package for creating PNG images.

[image] is the type of images being drawn. They can be created from scratch, imported from PNG files, and exported to PNG files.

All functions correctly clip to the actual size of the image.

[color] is the type of colors. Currently there can be at most 256 different colors in an image.

[style] is the type of drawing styles. A style is either a color, or transparent.

[mode] is the type of drawing modes for line drawing and filling. It may be one of

where c is a color Color c

Transparent

Brushed img for line drawing using the given image as brush for line drawing, cyclically using the styles Styled stys in the given vector to create a dashed line

StyledBrushed (vis, img)

for line drawing, using the given image as a brush, cyclically switching it on and off according to the

given bool vector

Tiled ima for filling, using the given image as a tile

[font] is the type of fonts: Tiny, Small, MediumBold, Large, Giant

[rgb] is the type of (r, g, b) triples, where the components indicate color intensity as an integer value in the range 0..255.

[xy] is the type of pairs, used for (x, y) coordinates and to indicate dimensions (width, height). The origin (0, 0) is the upper left-hand corner of the image. The x coordinates increase to the right; the y coordinates increase downwards.

[image (w, h) rgb] creates a new empty image with size (w, h) and the background color rgb. Raises Fail if the image cannot be

[fromPng filename] reads an image from the given PNG file. Raises Fail if the file does not exist or does not contain a PNG image.

[size img] returns (w, h) where w is the width and h the height of

[toPng img filename] write the image to the given file in PNG

[stdoutPng img] writes the image to standard output in PNG format, preceded by the HTTP header "Content-type: image/png\n\n". Useful in CGI scripts.

[color img rgb] returns the color code corresponding to rgb in the color table of img. Reuses the color code if it has already been allocated; otherwise allocates the color if possible; otherwise returns an approximation to the color rgb.

[htmlcolors im] returns a record containing the 16 standard HTML colors: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow. This call will allocate all these colors in the color table of the image, even if you do not use all of them.

[rgb img color] returns (r, g, b) where r, g, b are the component intensities of the given color in the color table of img.

[getTransparent img] returns SOME c where c is the 'transparent' color of the image, if any; otherwise returns NONE.

[setTransparent img col] makes the given color transparent in the

GDIMAGE

45

image.

[noTransparent img] makes all colors non-transparent in the image. This is useful for images that are to be used as tiles for filling. Such images are not allowed to have a transparent color.

[drawPixel img mode xy] draws the pixel in img at xy using the given mode.

[drawLine img mode (xy1, xy2)] draws a line in img from xy1 to xy2 using the given mode.

[drawRect img mode (xy1, xy2)] draws a rectangle in img with opposing corners xy1 and xy2 using the given mode.

[fillRect img mode (xy1, xy2)] draws a filled rectangle in img with opposing corners xy1 and xy2 using the given mode.

[drawPolygon img mode xys] draws a polygon in img with corners as given by the vector xys of coordinates using the given mode.

[fillPolygon img mode xys] draws a filled polygon in img with corners as given by the vector xys of coordinates using the given mode

[drawArc img mode { c, wh, from, to }] draw part of an ellipsis arc in img, with center c, width and height wh, using the given 'from' and 'to' angles, given in degrees (0..360).

[fill img mode xy] fills the region in img around xy which has the same color as the point at img, using the given mode.

[fillBorder img mode xy col] fills the region in img around xy which is delimited by the color col, using the given mode.

[copy { src, srcxy, srcwh, dst, dstxy }] copies part of the image src into the image dst, without rescaling. More precisely, copies the subimage of src whose upper left-hand corner is srcxy and whose size is srcwh, into the subimage of dst whose upper left-hand corner is dstxy. The images src and dst may be the same, but if the subimages overlap, then the result is unpredictable.

[copyResize { src, srcxy, srcwh, dst, dstxy, dstwh }] copies part of the image src into the image dst, rescaling to the given size dstwh of the destination subimage. Otherwise works as copy.

[char img col font xy ch] draws the character ch left-right (to be read from south) in img at xy using the given color.

[charUp img col font xy ch] draws the character ch bottom-up (to be read from east) in img at xy using the given color.

[string img col font xy s] draws the string s left-right (to be read from south) in img at xy using the given color.

[stringUp img col font xy s] draws the string s bottom-up (to be read from east) in img at xy using the given color.

[charsize font] returns (w, h) where w is the width and h the height, in pixels, of each character in the given font.

46 GENERAL

Module General

```
SML Basis Library and Moscow ML top-level declarations
SML Basis Library types
type
         exn
eqtype
        unit
datatype order = LESS | EQUAL | GREATER
Additional Moscow ML top-level types
datatype bool = false | true
eqtype char
eqtype int
datatype 'a option = NONE | SOME of 'a
type ppstream
eqtype real
eqtype string
type substring
type syserror
type 'a vector
eqtype word
eqtype word8
datatype 'a list = nil | op :: of 'a * 'a list
datatype 'a ref = ref of 'a
datatype 'a frag = QUOTE of string | ANTIQUOTE of 'a
SML Basis Library exceptions
exception Bind
exception Chr
exception Div
exception Domain
exception Fail of string
exception Match
exception Overflow
exception Option
exception Subscript
exception Size
Additional Moscow ML top-level exceptions
exception Graphic of string
exception Interrupt
exception Invalid_argument of string
exception Io of {function : string, name : string, cause : exn }
exception Out_of_memory
exception SysErr of string * syserror option
SML Basis Library values
              : 'a ref -> 'a
val!
              : 'a ref * 'a -> unit
val :=
               : ('b -> 'c) * ('a -> 'b) -> ('a -> 'c)
             : 'a -> unit
: 'a * 'b -> 'a
val ignore
val before
              : exn -> string
val exnName
val exnMessage : exn -> string
Additional Moscow ML top-level values
val not
           : bool -> bool
           : string * string -> string
val ^
           : "a * "a -> bool
val =
val <>
          : "a * "a -> bool
val ceil
          : real -> int
                                       round towards plus infinity
val floor : real -> int
                                       round towards minus infinity
```

val real : int -> real
val round : real -> int
val trunc : real -> int equals Real.fromInt round to nearest even round towards zero val vector : 'a list -> 'a vector Below, numtxt is int, Word.word, Word8.word, real, char, string: : numtxt * numtxt -> bool val <= : numtxt * numtxt -> bool val > : numtxt * numtxt -> bool val >= : numtxt * numtxt -> bool val makestring : numtxt -> string Below, realint is int or real: val ~ : realint -> realint raises Overflow val abs : realint -> realint raises Overflow Below, num is int, Word.word, Word8.word, or real: : num * num -> num raises Overflow : num * num -> num raises Overflow val val * : num * num -> num raises Overflow val / : real * real -> real raises Div, Overflow Below, wordint is int, Word.word or Word8.word: val div : wordint * wordint -> wordint raises Div, Overflow
val mod : wordint * wordint -> wordint raises Div [exn] is the type of exceptions.

[unit] is the type containing the empty tuple () which equals the empty record $\{\ \}.$

[order] is used as the return type of comparison functions.

[bool] is the type of booleans: false and true. Equals Bool.bool.

[char] is the type of characters such as #"A". Equals Char.char.

[int] is the type of integers. Equals Int.int.

[option] is the type of optional values. Equals Option.option.

[ppstream] is the type of pretty-printing streams, see structure PP. Pretty-printers may be installed in the top-level by function Meta.installPP; see the Moscow ML Owner's Manual.

[real] is the type of floating-point numbers. Equals Real.real.

[string] is the type of character strings. Equals String.string.

[substring] is the type of substrings. Equals Substring.substring.

[syserror] is the abstract type of system error codes. Equals OS.syserror.

[vector] is the type of immutable vectors. Equals Vector.vector.

[word] is the type of unsigned words. Equals Word.word.

[word8] is the type of unsigned bytes. Equals Word8.word.

['a list] is the type of lists of elements of type 'a. Equals List.list.

['a ref] is the type of mutable references to values of type 'a.

48 GENERAL

['a frag] is the type of quotation fragments, resulting from the parsing of quotations ' ... ' and antiquotations. See the Moscow ML Owner's Manual.

 $\left[\text{Bind} \right]$ is the exception raised when the right-hand side value in a valbind does not match the left-hand side pattern.

[Chr] signals an attempt to produce an unrepresentable character.

[Div] signals an attempt to divide by zero.

[Domain] signals an attempt to apply a function outside its domain of definition; such as computing Math.sqrt(~1).

 $\mbox{[Fail]}$ signals the failure of some function, usually in the Moscow ML specific library structures.

[Match] signals the failure to match a value against the patterns in a case, handle, or function application.

[Option] is raised by Option.valOf when applied to NONE.

[Overflow] signals the attempt to compute an unrepresentable number.

[Subscript] signals the attempt to use an illegal index in an array, dynarray, list, string, substring, vector or weak array.

[Size] signals the attempt to create an array, string or vector that is too large for the implementation.

[Graphic] signals the failure of Graphics primitives (DOS only).

[Interrupt] signals user interrupt of the computation.

[Invalid_argument] signals the failure of a function in the runtime system.

[Io { function, name, cause }] signals the failure of an input/output operation (function) when operating on a file (name). The third field (cause) may give a reason for the failure.

[Out_of_memory] signals an attempt to create a data structure too large for the implementation, or the failure to extend the heap or stack.

[SysErr (msg, err)] signals a system error, described by msg. A system error code may be given by err. If so, it will usually hold that msg = OS.errorMsg err.

SML Basis Library values

[! rf] returns the value pointed to by reference rf.

[:=(rf, e)] evaluates rf and e, then makes the reference rf point to
the value of e. Since := has infix status, this is usually written
 rf := e

[o(f, g)] computes the functional composition of f and g, that is, fn x => f(g x). Since o has infix status, this is usually written f o g

[ignore e] evaluates e, discards its value, and returns (): unit.

[before(e1, e2)] evaluates e1, then evaluates e2, then returns the value of e1. Since before has infix status, this is usually written e1 before e2

[exnName exn] returns a name for the exception constructor in exn.
Never raises an exception itself. The name returned may be that of
any exception constructor aliasing with exn. For instance,
 let exception E1; exception E2 = E1 in exnName E2 end

may evaluate to "E1" or "E2".

[exnMessage exn] formats and returns a message corresponding to exception exn. For the exceptions defined in the SML Basis Library, the message will include the argument carried by the exception.

Additional Moscow ML top-level values

[not b] returns the logical negation of b.

- [^] is the string concatenation operator.
- [=] is the polymorphic equality predicate.
- [<>] is the polymorphic inequality predicate.

[ceil r] is the smallest integer >= r (rounds towards plus infinity). May raise Overflow.

[floor r] is the largest integer \leftarrow r (rounds towards minus infinity). May raise Overflow.

[real i] is the floating-point number representing integer i. Equivalent to Real.fromInt.

[round r] is the integer nearest to r, using the default rounding mode. May raise Overflow.

[trunc r] is the numerically largest integer between r and zero (rounds towards zero). May raise Overflow.

[vector [x1, ..., xn]] returns the vector #[x1, ..., xn].

```
[< (x1, x2)]
[<=(x1, x2)]
[> (x1, x2)]
[>=(x1, x2)]
```

These are the standard comparison operators for arguments of type int, Word.word, Word8.word, real, char or string.

[makestring v] returns a representation of value v as a string, for v of type int, Word.word, Word8.word, real, char or string.

 $[\ \sim \ x]$ is the numeric negation of x (which can be real or int). May raise Overflow.

[abs x] is the absolute value of x (which can be real or int). May raise Overflow.

```
[+ (e1, e2)]
[- (e1, e2)]
[* (e1, e2)]
```

These are the standard arithmetic operations for arguments of type int, Word.word, Word8.word, and real. They are unsigned in the case of Word.word and Word8.word. May raise Overflow.

[/ (e1, e2)] is the floating-point result of dividing e1 by e2. May raise Div and Overflow.

 $[\operatorname{div}(\text{el}, \, \text{e2})]$ is the integral quotient of dividing el by e2 for arguments of type int, Word.word, and Word8.word. See Int.div and Word.div for more details. May raise Div, Overflow.

[mod(el, e2)] is the remainder when dividing el by e2, for arguments of type int, Word.word, and Word8.word. See Int.mod and Word.mod for more details. May raise Div.

50 HASHSET

Module Hashset

Hashset -- sets implemented by hashtables

```
signature Hashset = sig
type 'item set
exception NotFound
                   : ('_item -> word) * ('_item * '_item -> bool) -> '_item set
: ('_item -> word) * ('_item * '_item -> bool) -> '_item
val empty
val singleton
                      -> '_item set
                   : '_item set * '_item -> bool
: '_item set * '_item -> '_item
: '_item set * '_item -> '_item option
val member
val retrieve
val peek
                   : '_item set * '_item -> unit
: '_item set * '_item list -> unit
: '_item set * '_item -> unit
val add
val addList
val delete
                   : '_item set -> bool
val isEmpty
                   : '_item set * '_item set -> bool
: '_item set * '_item set -> bool
val isSubset
val equal
                   : '_item set -> int
val numItems
                   : '_item set -> '_item list
val listItems
                   : ('_item -> unit) -> '_item set -> unit
val app
                   : ('_item * 'b -> 'b) -> 'b -> '_item set -> 'b
val fold
                   : ('_item -> bool) -> '_item set -> bool
: ('_item -> bool) -> '_item set -> bool
: ('_item -> bool) -> '_item set -> '_item option
: ('_item set -> '_item set
val all
val exists
val find
val copy
val hash
                  : ' item set -> word
                : 'a -> word
val polyHash
end
   ['item set] is the type of sets of elements of type 'item, with a
   given hash function and equality predicate.
   [empty (hash, equal)] creates a new empty set with the given hash
   function and equality predicate. It must hold that equal(x, y)
   implies hash x = hash y.
   [singleton (hash, equal) i] creates the singleton set containing i,
   with the given hash function and equality predicate.
   [member(s, i)] returns true if and only if i is in s.
   [retrieve(s, i)] returns i if it is in s; raises NotFound otherwise.
   [peek(s, i)] returns SOME i if i is in s; returns NONE otherwise.
   [add(s, i)] adds item i to set s.
   [addList(s, xs)] adds all items from the list xs to the set s.
    [delete(s, i)] removes item i from s. Raises NotFound if i is not in s.
    [isEmpty s] returns true if the set is empty; false otherwise.
    [equal(s1, s2)] returns true if and only if the two sets have the
   same elements.
    [isSubset(s1, s2)] returns true if and only if s1 is a subset of s2.
   [numItems s] returns the number of items in set s.
   [listItems s] returns a list of the items in set s, in some order.
```

HASHSET 51

[app f s] applies function f to the elements of s, in some order.

[fold f e s] applies the folding function f to the entries of the set in some order.

[find p s] returns SOME i, where i is an item in s which satisfies p, if one exists; otherwise returns NONE.

[hash s] returns the hashcode of the set, which is the sum of the hashcodes of its elements, as computed by the hash function given when the set was created.

[polyHash v] returns a system-defined hashcode for the value v. This pseudo-polymorphic hash function can be used together with the standard equality function (=) to create a Hashset for any type that admits equality, as follows:

val set = Hashset.empty (Hashset.hash, op =);

52 HELP

Module Help

Help -- on-line help functions : string -> unit val help : int ref val displayLines val helpdirs : string list ref val indexfiles : string list ref : {term : string, file : string, title : string} list ref val specialfiles : string vector ref val welcome val browser : (string -> unit) ref val defaultBrowser : string -> unit

[help s] provides on-line help on the topic indicated by string s.

```
help "lib"; gives an overview of the Moscow ML library.
help "id"; provides help on identifier id (case-insensitive).
```

If exactly one identifier in the library matches id (case-insensitive), then the browser opens the signature defining that identifier, positioning the first occurrence of id at the center of the screen.

If more than one identifier matches id (case-insensitive), then a small menu lists the signatures containing the identifier. To invoke the browser, just type in the number of the desired signature.

The browser accepts the following commands, which must be followed by a newline:

```
d    move down by half a screen
u    move up by half a screen
t    move to top of file
b    move to bottom of file
/str    cyclically search for string str in help file (case-insensitive)
n    search for next occurrence of str
q    quit the browser
```

A newline by itself moves down one screen (24 lines).

[helpdirs] is a reference to a list of additional directories to be searched for help files. The directories are searched in order, after the -stdlib directory.

[indexfiles] is a reference to a list of full paths of help term index files. Setting 'indexfiles' affects subsequent invocations of 'help'. (Every invocation of 'help' reads the index files anew).

[specialfiles] is a reference to a list of {term, file, title} records, each of which maps a search term to the specified file with the specified title (in the browser). The string in the 'term' field should be all lowercase, since the argument passed to 'help' will be converted to lowercase.

[welcome] is a reference to the text shown in response to the query help "". This is a vector of lines of text.

[browser] is a reference to the function that gets invoked on the text of the help file. Initially set to defaultBrowser.

[defaultBrowser] is the default (built-in) help browser.

[displayLines] is a reference to the size of the display (window) assumed by the defaultBrowser; initially 24 lines. Set it to the actual size of your window for best results.

INT 53

Module Int

```
Int -- SML Basis Library
type int = int
val precision : int option
            : int option
val minInt
val maxInt
             : int option
val ~
             : int -> int
                                        Overflow
val *
             : int * int -> int
                                        Overflow
             : int * int -> int
val div
                                        Div, Overflow
             : int * int -> int
                                        Div
val mod
             : int * int -> int
                                        Div, Overflow
val quot
             : int * int -> int
: int * int -> int
val rem
                                        Div
                                        Overflow
val +
             : int * int -> int
                                        Overflow
val -
             : int * int -> bool
val >
             : int * int -> bool
val >=
             : int * int -> bool
val <
             : int * int -> bool
val <=
            : int -> int
: int * int -> int
                                        Overflow
val abs
val min
             : int * int -> int
val max
             : int -> int
val sign
val sameSign : int * int -> bool
             : int * int -> order
val compare
val toInt
             : int -> int
val fromInt    : int -> int
val toLarge    : int -> int
val fromLarge    : int -> int
val scan
              : StringCvt.radix
                -> (char, 'a) StringCvt.reader -> (int, 'a) StringCvt.reader
val fmt
              : StringCvt.radix -> int -> string
val toString : int -> string
[precision] is SOME n, where n is the number of significant bits in an
   integer. In Moscow ML n is 31 in 32-bit architectures and 63 in 64-bit
   architectures.
   [minInt] is SOME n, where n is the most negative integer.
```

```
[maxInt] is SOME n, where n is the most positive integer.
[*]
[+]
[\,\text{-}\,] are the usual operations on integers. They raise Overflow if
the result is not representable as an integer.
[abs] returns the absolute value of its argument. Raises Overflow
if applied to the most negative integer.
[div] is integer division, rounding towards minus infinity.
Evaluating i div 0 raises Div. Evaluating i div ~1 raises
Overflow if i is the most negative integer.
[mod] is the remainder for div. If q = i \text{ div } d and r = i \text{ mod } d then
it holds that qd + r = i, where either 0 <= r < d or d < r <= 0.
Evaluating i mod 0 raises Div, whereas i mod ~1 = 0, for all i.
[quot] is integer division, rounding towards zero. Evaluating
quot(i, 0) raises Div. Evaluating quot(i, ~1) raises Overflow if i
is the most negative integer.
```

54 INT

[rem(i, d)] is the remainder for quot. That is, if q = quot(i, d) and r = rem(i, d) then d * q + r = i, where r is zero or has the same sign as i. If made infix, the recommended fixity for quot and rem is

infix 7 quot rem

 $[\min(x, y)]$ is the smaller of x and y.

 $[\max(x, y)]$ is the larger of x and y.

[sign x] is ~ 1 , 0, or 1, according as x is negative, zero, or positive.

[<] [<=]

[>]

[>=] are the usual comparisons on integers.

[compare(x, y)] returns LESS, EQUAL, or GREATER, according as x is less than, equal to, or greater than y.

[sameSign(x, y)] is true iff sign x = sign y.

[toInt x] is x (because this is the default int type in Moscow ML).

[fromInt x] is x (because this is the default int type in Moscow ML).

[toLarge x] is x (because this is the largest int type in Moscow ML).

[fromLarge x] is x (because this is the largest int type in Moscow ML).

[fmt radix i] returns a string representing i, in the radix (base) specified by radix.

radix	description		output format
BIN	signed binary	(base 2)	~?[01]+
OCT	signed octal	(base 8)	~?[0-7]+
DEC	signed decimal	(base 10)	~?[0-9]+
HEX	signed hexadecimal	(base 16)	~?[0-9A-F]+

[toString i] returns a string representing i in signed decimal format. Equivalent to (fmt DEC i).

[fromString s] returns SOME(i) if a decimal integer numeral can be scanned from a prefix of string s, ignoring any initial whitespace; returns NONE otherwise. A decimal integer numeral must have form, after possible initial whitespace: $[+\sim-]?[0-9]+$

[scan radix getc charsrc] attempts to scan an integer numeral from the character source charsrc, using the accessor getc, and ignoring any initial whitespace. The radix argument specifies the base of the numeral (BIN, OCT, DEC, HEX). If successful, it returns SOME(i, rest) where i is the value of the number scanned, and rest is the unused part of the character source. A numeral must have form, after possible initial whitespace:

```
radix input format
-----
BIN [+~-]?[0-1]+
OCT [+~-]?[0-7]+
DEC [+~-]?[0-9]+
HEX [+~-]?[0-9a-fA-F]+
```

INTMAP 55

Module Intmap

Intmap -- Applicative maps with integer keys

From SML/NJ lib 0.2, copyright 1993 by AT&T Bell Laboratories Original implementation due to Stephen Adams, Southampton, UK

```
type 'a intmap
exception NotFound
              : unit -> 'a intmap
val empty
               : 'a intmap * int * 'a -> 'a intmap
val insert
val retrieve : 'a intmap * int -> 'a
               : 'a intmap * int -> 'a option
val peek
               : 'a intmap * int -> 'a intmap * 'a
val remove
val numItems : 'a intmap -> int
val listItems : 'a intmap -> (int * 'a) list
               : (int * 'a -> unit) -> 'a intmap -> unit
val app
               : (int * 'a -> unit) -> 'a intmap -> unit
: (int * 'a * 'b -> 'b) -> 'b -> 'a intmap -> 'b
val revapp
val foldr
              : (int * 'a * 'b -> 'b) -> 'b -> 'a intmap -> 'b
val foldl
val map : (int * 'a -> 'b) -> 'a intmap -> 'b intmap val transform : ('a -> 'b) -> 'a intmap -> 'b intmap
   ['a intmap] is the type of applicative maps from int to 'a.
   [empty] creates a new empty map.
   [insert(m, i, v)] extends (or modifies) map m to map i to v.
   [retrieve(m, i)] returns v if m maps i to v; otherwise raises
   [peek(m, i)] returns SOME v if m maps i to v; otherwise NONE.
   [remove(m, i)] removes i from the domain of m and returns the
   modified map and the element v corresponding to i. Raises NotFound
   if i is not in the domain of m.
   [numItems m] returns the number of entries in m (that is, the size
   of the domain of m).
   [listItems m] returns a list of the entries (i, v) of integers i and
   the corresponding values v in m, in increasing order of i.
   [app f m] applies function f to the entries (i, v) in m, in
   increasing order of i.
   [revapp f m] applies function f to the entries (i, v) in m, in
   decreasing order of i.
   [foldl f e m] applies the folding function f to the entries (i, v)
   in m, in increasing order of i.
   [foldr f e m] applies the folding function f to the entries (i, v)
   in m, in decreasing order of i.
   [map f m] returns a new map whose entries have form (i, f(i,v)),
   where (i, v) is an entry in m.
   [transform f m] returns a new map whose entries have form (i, f(i,v)),
   where (i, v) is an entry in m.
```

56 INTSET

Module Intset

```
Intset -- applicative sets of integers
From SML/NJ lib 0.2, copyright 1993 by AT&T Bell Laboratories
Original implementation due to Stephen Adams, Southampton, UK
type intset
exception NotFound
val empty
                 : intset
val singleton
                 : int -> intset
val add
                 : intset * int -> intset
                 : intset * int list -> intset
val addList
val isEmpty
                 : intset -> bool
                 : intset * intset -> bool
val equal
val isSubset
                 : intset * intset -> bool
                 : intset * int -> bool
val member
                 : intset * int -> intset
val delete
val numItems
                 : intset ->
                              int
                 : intset * intset -> intset
val union
val intersection : intset * intset -> intset
                : intset * intset -> intset
val difference
val listItems
                 : intset -> int list
                 : (int -> unit) -> intset -> unit
val app
                 : (int -> unit) -> intset -> unit
: (int * 'b -> 'b) -> 'b -> intset -> 'b
val revapp
val foldr
                 : (int * 'b -> 'b) -> 'b -> intset -> 'b
val foldl
val find
                 : (int -> bool) -> intset -> int option
   [intset] is the type of sets of integers.
   [empty] is the empty set of integers.
   [singleton i] is the singleton set containing i.
   [add(s, i)] adds item i to set s.
   [addList(s, xs)] adds all items from the list xs to the set s.
   [isEmpty s] returns true if and only if the set is empty.
   [equal(s1, s2)] returns true if and only if the two sets have the
   same elements.
   [isSubset(s1, s2)] returns true if and only if s1 is a subset of s2.
   [member(s, i)] returns true if and only if i is in s.
   [delete(s, i)] removes item i from s. Raises NotFound if i is not in s.
   [numItems s] returns the number of items in set s.
   [union(s1, s2)] returns the union of s1 and s2.
   [intersection(s1, s2)] returns the intersection of s1 and s2.
   [difference(s1, s2)] returns the difference between s1 and s2 (that
   is, the set of elements in s1 but not in s2).
   [listItems s] returns a list of the items in set s, in increasing
   order.
   [app f s] applies function f to the elements of s, in increasing
   order.
   [revapp f s] applies function f to the elements of s, in decreasing
   order.
   [foldl f e s] applies the folding function f to the entries of the
   set in increasing order.
```

INTSET 57

[foldr f e s] applies the folding function f to the entries of the set in decreasing order.

[find p s] returns SOME i, where i is an item in s which satisfies p, if one exists; otherwise returns $\ensuremath{\mathtt{NONE}}$.

58 LEXING

Module Lexing

Lexing -- run-time library for lexers generated by mosmllex Closely based on the library for camllex. Copyright 1993 INRIA, France local open Obj in

type lexbuf val createLexerString : string -> lexbuf
val createLexer : (CharArray.array -> int -> int) -> lexbuf : lexbuf -> string val getLexeme val getLexemeChar : lexbuf -> int -> char : lexbuf -> int val getLexemeStart val getLexemeEnd : lexbuf -> int For internal use in generated lexers: : lexbuf -> obj val dummyAction val backtrack : lexbuf -> 'a prim_val getNextChar : lexbuf -> char = 1 "get_next_char"

end

These functions are for use in mosmllex-generated lexers. For further information, see the Moscow ML Owner's Manual. For examples, see mosml/examples/lexyacc and mosml/examples/calc.

[lexbuf] is the type of lexer buffers. A lexer buffer is the argument passed to the scanning functions defined by the mosmllex-generated scanners. The lexer buffer holds the current state of the scanner, plus a function to refill the buffer from the input.

[createLexerString s] returns a lexer buffer which reads from the given string s. Reading starts from the first character in the string. An end-of-input condition is generated when the end of the string is reached.

[createLexer f] returns a lexer buffer that will use the given function f for reading additional input. When the lexer needs more characters, it will call the given function as (f carr n), where carr is a character array, and n is an integer. The function should put at most characters or in carr, starting at character number 0, and return the number of characters actually stored. A return value of 0 means end of input.

A lexer definition (input to mosmllex) consists of fragments of this form

```
parse
    lhs1 { rhs1 }
    lhs2 { rhs2 }
    lhs3 { rhs3 }
    ...
```

where the lhs are regular expressions matching some string of

LEXING 59

characters, and the rhs are corresponding semantic actions, written in ML. The following functions can be used in the semantic actions:

[getLexeme lexbuf] returns the string matched by the left-hand side regular expression.

[getLexemeChar lexbuf i] returns character number i in the matched string.

[getLexemeStart lexbuf] returns the start position of the matched string (in the input stream). The first character in the stream has position $\bf 0$.

[getLexemeEnd lexbuf] returns the end position, plus one, of the matched string (in the input stream). The first character in the stream has position 0.

60 LIST

Module List

```
List -- SML Basis Library
datatype list = datatype list
exception Empty
                : 'a list -> bool
val null
                : 'a list -> 'a
val hd
                                                             Empty
               : 'a list -> 'a list
val tl
                                                             Empty
               : 'a list -> 'a
val last
                                                             Empty
                : 'a list * int -> 'a
val nth
                                                             Subscript
                : 'a list * int -> 'a list
val take
                                                             Subscript
                : 'a list * int -> 'a list
val drop
                                                             Subscript
               : 'a list -> int
val length
val rev
                : 'a list -> 'a list
val @ : 'a list * 'a list -> 'a list
val concat : 'a list list -> 'a list
val revAppend : 'a list * 'a list -> 'a list
              : ('a -> unit) -> 'a list -> unit
: ('a -> 'b) -> 'a list -> 'b list
val app
val map
val mapPartial : ('a -> 'b option) -> 'a list -> 'b list
             : ('a -> bool) -> 'a list -> 'a option
: ('a -> bool) -> 'a list -> 'a list
val find
val filter
val partition : ('a -> bool ) -> 'a list -> ('a list * 'a list)
               : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
: ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
val foldr
val foldl
val exists
               : ('a -> bool) -> 'a list -> bool
                : ('a -> bool) -> 'a list -> bool
val all
val collate
                : ('a * 'a -> order) -> 'a list * 'a list -> order
val tabulate : int * (int -> 'a) -> 'a list
                                                             Size
              : 'a list -> ('a * 'a list) option
val getItem
   ['a list] is the type of lists of elements of type 'a.
   [null xs] is true iff xs is nil.
   [hd xs] returns the first element of xs. Raises Empty if xs is nil.
   [tl xs] returns all but the first element of xs.
   Raises Empty if xs is nil.
   [last xs] returns the last element of xs. Raises Empty if xs is nil.
   [nth(xs, i)] returns the i'th element of xs, counting from 0.
   Raises Subscript if i<0 or i>=length xs.
   [take(xs, i)] returns the first i elements of xs. Raises Subscript
   if i<0 or i>length xs.
   [drop(xs, i)] returns what is left after dropping the first i
   elements of xs. Raises Subscript if i<0 or i>length xs.
   It holds that take(xs, i) @ drop(xs, i) = xs when 0 <= i <= length xs.
   [length xs] returns the number of elements in xs.
   [rev xs] returns the list of xs's elements, reversed.
```

[xs @ ys] returns the list which is the concatenation of xs and ys.

[concat xss] returns the list which is the concatenation of all the lists in xss.

[revAppend(xs, ys)] is equivalent to rev xs @ ys, but more efficient.

[app f xs] applies f to the elements of xs, from left to right.

[map f xs] applies f to each element x of xs, from left to right, and returns the list of f's results.

[mapPartial f xs] applies f to each element x of xs, from left to right, and returns the list of those y's for which f(x) evaluated to SOME y.

[find p xs] applies p to each element x of xs, from left to right, until p(x) evaluates to true; returns SOME x if such an x exists, otherwise NONE.

[filter p xs] applies p to each element x of xs, from left to right, and returns the sublist of those x for which p(x) evaluated to true.

[partition p xs] applies p to each element x of xs, from left to right, and returns a pair (pos, neg) where pos is the sublist of those x for which p(x) evaluated to true, and neg is the sublist of those for which p(x) evaluated to false.

[foldr op% e xs] evaluates x1 % (x2 % (... % (x(n-1) % (xn % e)) ...)) where xs = [x1, x2, ..., x(n-1), xn], and % is taken to be infixed.

[foldl op% e xs] evaluates xn % (x(n-1) % (... % (x2 % (x1 % e)))) where xs = [x1, x2, ..., x(n-1), xn], and % is taken to be infixed.

[exists p xs] applies p to each element x of xs, from left to right until p(x) evaluates to true; returns true if such an x exists, otherwise false.

[all p xs] applies p to each element x of xs, from left to right until p(x) evaluates to false; returns false if such an x exists, otherwise true.

[collate cmp (xs, ys)] returns LESS, EQUAL or GREATER according as xs precedes, equals or follows ys in the lexicographic ordering on lists induced by the ordering cmp on elements.

[tabulate(n, f)] returns a list of length n whose elements are f(0), f(1), ..., f(n-1), created from left to right. Raises Size if n<0.

[getItem xs] attempts to extract an element from the list xs. It returns NONE if xs is empty, and returns SOME (x, xr) if xs=x::xr. This can be used for scanning booleans, integers, reals, and so on from a list of characters. For instance, to scan a decimal integer from a list cs of characters, compute

Int.scan StringCvt.DEC List.getItem cs

62 LISTPAIR

Module ListPair

```
ListPair -- SML Basis Library
              : 'a list * 'b list -> ('a * 'b) list
val zip
val unzip : ('a * 'b) list -> 'a list * 'b list
val map : ('a * 'b -> 'c) -> 'a list * 'b list -> 'c list
              : ('a * 'b -> unit) -> 'a list * 'b list -> unit
val all : ('a * 'b -> bool) -> 'a list * 'b list -> bool
val exists : ('a * 'b -> bool) -> 'a list * 'b list -> bool
val foldr : ('a * 'b * 'c -> 'c) -> 'c -> 'a list * 'b list -> 'c
val foldl : ('a * 'b * 'c -> 'c) -> 'c -> 'a list * 'b list -> 'c
               : ('a * 'b -> bool) -> 'a list * 'b list -> bool
val allEq
exception UnequalLengths
                : ('a list * 'b list) -> ('a * 'b) list
val zipEq
                : ('a * 'b -> 'c) -> 'a list * 'b list -> 'c list
: ('a * 'b -> 'c) -> 'a list * 'b list -> unit
val mapEq
val appEq
val foldrEq : ('a * 'b * 'c -> 'c) -> 'c -> 'a list * 'b list -> 'c
val foldlEq : ('a * 'b * 'c -> 'c) -> 'c -> 'a list * 'b list -> 'c
```

These functions process pairs (xs, ys) of lists. There are three groups of functions:

- * zip, map, app, all, exists, foldr and foldl raise no exception when the argument lists are found to be of unequal length; the excess elements from the longer list are simply disregarded.
- * zipEq, mapEq, appEq, foldrEq and foldlEq raise exception UnequalLengths when the argument lists are found to be of unequal length.
- * allEq raises no exception but returns false if the lists are found to have unequal lengths (after traversing the lists).

 $[{\tt zip}\ ({\tt xs},\ {\tt ys})]$ returns the list of pairs of corresponding elements from ${\tt xs}$ and ${\tt ys}.$

[unzip xys] returns a pair (xs, ys), where xs is the list of first components of xys, and ys is the list of second components from xys. Hence zip (unzip xys) has the same result and effect as xys.

[map f (xs, ys)] applies function f to the pairs of corresponding elements of xs and ys from left to right and returns the list of results. Hence map f (xs, ys) has the same result and effect as List.map f (zip (xs, ys)).

[app f (xs, ys)] applies function f to the pairs of corresponding elements of xs and ys from left to right and returns (). Hence app f (xs, ys) has the same result and effect as List.app f (zip (xs, ys)).

[all p (xs, ys)] applies predicate p to the pairs of corresponding elements of xs and ys from left to right until p evaluates to false or one or both lists is exhausted; returns true if p is true of all such pairs; otherwise false. Hence all p (xs, ys) has the same result and effect as List.all p (zip (xs, ys)).

[exists p (xs, ys)] applies predicate p to the pairs of corresponding elements of xs and ys from left to right until p evaluates to true or one or both lists is exhausted; returns true if p is true of any such pair; otherwise false. Hence exists p (xs, ys) has the same result and effect as List.exists p (zip (xs, ys)). Also, exists p (xs, ys) is equivalent to not(all (not o p) (xs, ys)).

```
[foldr f e (xs, ys)] evaluates f(x1, y1, f(x2, y2, f(..., f(xn, yn, e)))) where xs = [x1, x2, ..., x(n-1), xn, ...], ys = [y1, y2, ..., y(n-1), yn, ...],
```

LISTPAIR 63

```
n = min(length xs, length ys).
Equivalent to List.foldr (fn ((x, y), r) \Rightarrow f(x, y, r)) e (zip(xs, ys)).
[foldl f e (xs, ys)] evaluates f(xn, yn, f(..., f(x2, y2, f(x1, y1, e))))
where xs = [x1, x2, ..., x(n-1), xn, ...],
ys = [y1, y2, \dots, y(n-1), yn, \dots], and n = min(length xs, length ys). Equivalent to List.foldl (fn ((x, y), r) => f(x, y, r)) e (zip(xs, ys)).
[zipEq (xs, ys)] returns the list of pairs of corresponding
elements from xs and ys. Raises UnequalLengths if xs and ys do not
have the same length.
[mapEq f (xs, ys)] applies function f to pairs of corresponding
elements of xs and ys from left to right, and then returns the list
of results if xs and ys have the same length, otherwise raises UnequalLengths. If f has no side effects and terminates, then
it is equivalent to List.map f (zipEq (xs, ys)).
[appEq f (xs, ys)] applies function f to pairs of corresponding
elements of xs and ys from left to right, and then raises
UnequalLengths if xs and ys have the same length.
[foldrEq f e (xs, ys)] raises UnequalLengths if xs and ys do not
have the same length. Otherwise evaluates
      f(x1, y1, f(x2, y2, f(..., f(xn, yn, e))))
where xs = [x1, x2, ..., x(n-1), xn],
ys = [y1, y2, ..., y(n-1), yn],
and n = length xs = length ys.
Equivalent to List.foldr (fn ((x,y),r) \Rightarrow f(x,y,r)) e (zipEq(xs, ys)).
[foldlEq f e (xs, ys)] evaluates f(xn, yn, f(..., f(x2, y2, f(x1, y1, e)))) where xs = [x1, x2, ..., x(n-1), xn, ...],
      ys = [y1, y2, ..., y(n-1), yn, ...],
       n = min(length xs, length ys).
Then raises UnequalLengths if xs and ys do not have the same
length. If f has no side effects and terminates normally, then it is
equivalent to List.foldl (fn ((x,y),r) = f(x,y,r)) e (zipEq(xs, ys)).
[allEq p (xs, ys)] works as all p (xs, ys) but returns false if xs
and ys do not have the same length. Equivalent to
    all p (xs, ys) and also length xs = length ys.
```

64 LISTSORT

Module Listsort

```
Listsort
```

```
val sort : ('a * 'a -> order) -> 'a list -> 'a list
val sorted : ('a * 'a -> order) -> 'a list -> bool
val merge : ('a * 'a -> order) -> 'a list * 'a list -> 'a list
val mergeUniq : ('a * 'a -> order) -> 'a list * 'a list -> 'a list
val eqclasses : ('a * 'a -> order) -> 'a list -> 'a list list
```

[sort ordr xs] sorts the list xs in nondecreasing order, using the given ordering. Uses Richard O'Keefe's smooth applicative merge sort.

[sorted ordr xs] checks that the list xs is sorted in nondecreasing order, in the given ordering.

[merge ordr (xs, ys)] returns a sorted list of the elements of the sorted lists xs and ys, preserving duplicates. Both xs and ys must be sorted by ordr, that is, must satisfy sorted ordr xs andalso sorted ordr ys Then the result satisfies sorted ordr (merge ordr (xs, ys))

[mergeUniq ordr (xs, ys)] returns a sorted list of the elements of the sorted lists xs and ys, without duplicates. Both xs and ys must be sorted by ordr.

[eqclasses ordr xs] returns a list [xs1, xs2, ..., xsn] of
non-empty equivalence classes of xs, obtained by sorting the list
and then grouping consecutive runs of elements that are EQUAL by ordr.
If ordr is a total order, then it holds for xi in xsi and xj in xsj:
 ordr(xi, xj) = EQUAL iff i=j and
 ordr(xi, xj) = LESS iff i<j and
 ordr(xi, xj) = GREATER iff i>j
Thus ordr(xi, xj) = Int.compare(i, j). A list of representatives
for the equivalence classes of xs under ordering ordr can be
obtained by
 List.map List.hd (eqclasses ordr xs)

LOCATION 65

Module Location

Location -- error reporting for mosmllex and mosmlyac Based on src/compiler/location from the Caml Light 0.6 distribution

val errLocation : string * BasicIO.instream * Lexing.lexbuf -> Location

-> unit

: string * BasicIO.instream * Lexing.lexbuf -> Location val errMsq

-> string -> 'a

val errPrompt : string -> unit;

val nilLocation : Location

val getCurrentLocation : unit -> Location
val mkLoc : 'a -> Location * 'a val xLR : Location * 'a -> Location : Location * 'a -> int val xL

val xR : Location * 'a -> int val xxLR : Location * 'a -> Location * 'b -> Location val xxRL : Location * 'a -> Location * 'b -> Location

These functions support error reporting in lexers and parsers generated with mosmllex and mosmlyac. The directory mosml/examples/lexyacc/ contains an example of their use.

[errLocation (file, stream, lexbuf) loc] prints the part of the lexer input which is indicated by location loc.

If file <> "" then it is assumed to be the name of the file from which the lexer reads, the stream is assumed to be an open input stream associated with this file, and lexbuf is the lexer buffer used to read from the stream. Under MS DOS (and presumably Windows, OS/2, and MacOS), the stream must have been opened in binary mode (with Nonstdio.open_in_bin), or else the positioning in the file will be wrong (due to the translation of CRLF into newline in text files).

If file = "" then the lexer is assumed to read from some source $\[$ other than a stream, and the lexbuf (rather than the instream) is used to obtain the location indicated, if possible. In this case the stream is immaterial; it will not be used.

[errMsg (file, stream, lexbuf) loc msg] calls errLocation to print the indicated part of the lexer input, then prints the error $\frac{1}{2}$ message msg and raises exception Fail.

[errPrompt msg] prints "! ", the string msg, and a newline on standard output.

[nilLocation] is the undefined location.

[getCurrentLocation ()] can be called within the semantic action part of a grammar rule (only) and returns the location of the string matching the left-hand side of the rule.

[mkLoc a] can be called within the semantic action part of a grammar rule (only), and returns a pair (loc, a) of the current location and the value a. This is typically used to decorate abstract syntax tree nodes with location information, for use in subsequent error reports.

[xLR loc_a] returns the location of the decorated value loc_a.

[xL loc a] returns the left end position of loc a.

[xR loc_a] returns the right end position of loc_a.

[xxLR loc_a loc_b] returns the location extending from the left end of loc_a to the right end of loc_b.

66 LOCATION

[xxRL loc_a loc_b] returns the location extending from the right end of loc_a to the left end of loc_b.

MATH 67

Module Math

type real = real

Math -- SML Basis Library

```
val pi
          : real
val e
         : real
val sqrt : real -> real
         : real -> real
val sin
         : real -> real
val cos
val tan
          : real -> real
val atan : real -> real
val asin : real -> real
val acos : real -> real
val atan2 : real * real -> real
val exp : real -> real
         : real * real -> real
val pow
val ln
          : real -> real
val log10 : real -> real
val sinh : real -> real
val cosh : real -> real
val tanh : real -> real
   [pi] is the circumference of the circle with diameter 1, that is,
   3.14159265358979323846.
   [e] is the base of the natural logarithm: 2.7182818284590452354.
   [sqrt x] is the square root of x. Raises Domain if x < 0.0.
   [sin r] is the sine of r, where r is in radians.
   [cos r] is the cosine of r, where r is in radians.
   [tan r] is the tangent of r, where r is in radians. Raises Domain if
   r is a multiple of pi/2.
   [atan t] is the arc tangent of t, in the open interval ] ~pi/2, pi/2 [.
   [asin t] is the arc sine of t, in the closed interval [ \simpi/2, pi/2 ].
   Raises Domain if abs x > 1.
   [acos t] is the arc cosine of t, in the closed interval [ 0, pi ].
   Raises Domain if abs x > 1.
   [atan2(y, x)] is the arc tangent of y/x, in the interval ] \simpi, pi ],
   except that atan2(y, 0) = sign y * pi/2. The quadrant of the result
   is the same as the quadrant of the point (x, y).
   Hence sign(cos(atan2(y, x))) = sign x
   and sign(sin(atan2(y, x))) = sign y.
   [\exp x] is e to the x'th power.
   [pow (x, y)] is x it the y'th power, defined when y \ge 0 and (y integral or x >= 0)
   or y < 0 and ((y integral and x <> 0.0) or x > 0).
   We define pow(0, 0) = 1.
   [\ln x] is the natural logarithm of x (that is, with base e).
   Raises Domain if x \le 0.0.
   [log10 x] is the base-10 logarithm of x. Raises Domain if x \le 0.0.
   [\sinh x] returns the hyperbolic sine of x, mathematically defined as
   (\exp x - \exp (\sim x)) / 2. Raises Overflow if x is too large.
   [cosh x] returns the hyperbolic cosine of x, mathematically defined as
   (\exp x + \exp (\sim x)) / 2. Raises Overflow if x is too large.
```

68 MATH

[tanh x] returns the hyperbolic tangent of x, mathematically defined as (sinh x) / (cosh x). Raises Domain if x is too large.

META 69

Module Meta

Meta -- functions available only in interactive Moscow ML sessions

val printVal : 'a -> 'a
val printDepth : int ref
val printLength : int ref
val installPP : (ppstream -> 'a -> unit) -> unit

val liberal : unit -> unit

val orthodox : unit -> unit
val use : string -> unit
val compile : string -> unit

val conservative : unit -> unit

val quit : unit -> 'a

These values and functions are available in the Moscow ML interactive system only.

[printVal e] prints the value of expression e to standard output exactly as it would be printed at top-level, and returns the value of e. Output is flushed immediately. This function is provided as a simple debugging aid. The effect of printVal is similar to that of 'print' in Edinburgh ML or Umeaa ML. For string arguments, the effect of SML/NJ print can be achieved by the function TextIO.print: string -> unit.

[quit ()] quits Moscow ML immediately.

[installPP pp] installs the prettyprinter pp: ppstream -> ty -> unit at type ty. The type ty must be a nullary (parameter-less) type constructor representing a datatype, either built-in (such as bool) or user-defined. Whenever a value of type ty is about to be printed by the interactive system, or function printVal is invoked on an argument of type ty, the pretty-printer pp will be invoked to print it. See library unit PP for more information.

[use "f"] causes ML declarations to be read from file f as if they were entered from the console. A file loaded by use may, in turn, evaluate calls to use. For best results, use 'use' only at top

70 META

level, or at top level within a use'd file.

[liberal ()] sets liberal mode for the compilation functions: accept (without warnings) all extensions to the SML Modules language. The extensions are: higher-order modules (functors defined within structures and functors); first-order modules (structures can be packed as values, and values can be unpacked as structures); and recursively defined modules (signatures and structures). The liberal, conservative, and orthodox modes affect the functions compile, compileStructure, and compileToplevel. The liberal mode may be set also by the mosml option -liberal.

[conservative ()] sets conservative mode for the compilation functions: accept all extensions to the SML Modules language, but issue a warning for each use. The conservative mode may be set also by the mosml option -conservative. This is the default.

[orthodox ()] sets orthodox mode for the compilation functions: reject all uses of the extensions to the SML Modules language. That is, accept only SML Modules syntax. The orthodox mode may be set also by the mosml option -orthodox.

[compile "U.sig"] will compile and elaborate the specifications in file U.sig in structure mode, producing a compiled signature U in file U.ui. This function is backwards compatible with Moscow ML 1.44 and earlier. Equivalent to compileStructure [] "U.sig".

[compile "U.sml"] will elaborate and compile the declarations in file U.sml in structure mode, producing a compiled structure U in bytecode file U.uo. If there is an explicit signature file U.sig, then file U.ui must exist, and the unit body must match the signature. If there is no U.sig, then an inferred signature file U.ui will be produced also. No evaluation takes place. This function is backwards compatible with Moscow ML 1.44 and earlier. Equivalent to compileStructure [] "U.sml".

The declared identifiers will be reported if verbose is true (see below); otherwise compilation will be silent. In any case, compilation warnings are reported, and compilation errors abort the compilation and raise the exception Fail with a string argument.

[compileStructure opnunits "U.sml"] compiles the declarations in
file U.sml as if they formed a structure declaration
 structure U = struct ... contents of U.sml ... end
The contents of opnunits is added to the compilation context in
which the declarations in U.sml are compiled. If U.ui exists
already and represents a signature called U, then the compiled
declarations are matched against it. The result is a bytecode file
U.uo. If no file U.ui existed, then also a file U.ui is created,
containing the inferred signature of structure U. This
corresponds to invoking the batch compiler as follows:
 mosmlc -c U1.ui ... Un.ui -structure U.sml
where opnunits equals ["U1", ..., "Un"].

[compileToplevel opnunits "U.sig"] compiles the specifications in file U.sig, in a context in which all declarations from opnunits are visible, creating a compiled signature file U.ui. This corresponds to invoking the batch compiler as follows: mosmlc -c U1.ui ... Un.ui -toplevel U.sig where opnunits equals ["U1", ..., "Un"].

[compileToplevel opnunits "U.sml"] compiles the declarations in

META 71

file U.sml, in a context in which all declarations from opnunits are visible, creating a bytecode file U.uo. If U.ui exists already, then the compiled declarations are matched against it; otherwise the file U.ui is created. This corresponds to invoking the batch compiler as follows

mosmlc -c U1.ui ... Un.ui -toplevel U.sml
where opnunits equals ["U1", ..., "Un"].

[load "U"] will load and evaluate the compiled unit body from file U.uo. The resulting values are not reported, but exceptions are reported, and cause evaluation and loading to stop. If U is already loaded, then load "U" has no effect. If any other unit is mentioned by U but not yet loaded, then it will be loaded automatically before U.

After loading a unit, it can be opened with 'open U'. Opening it at top-level will list the identifiers declared in the unit.

When loading U, it is checked that the signatures of units mentioned by U agree with the signatures used when compiling U, and it is checked that the signature of U has not been modified since U was compiled; these checks are necessary for type safety. The exception Fail is raised if these signature checks fail, or if the file containing U or a unit mentioned by U does not exist.

[loadOne "U"] is similar to 'load "U"', but raises exception Fail if U is already loaded or if some unit mentioned by U is not yet loaded. That is, it does not automatically load any units mentioned by U. It performs the same signature checks as 'load'.

[loaded ()] returns a list of the names of all compiled units that have been loaded so far. The names appear in some random order.

[loadPath] determines the load path: which directories will be searched for interface files (.ui files), bytecode files (.uo files), and source files (.sml files). This variable affects the load, loadOne, and use functions. The current directory is always searched first, followed by the directories in loadPath, in order. By default, only the standard library directory is in the list, but if additional directories are specified using option -I, then these directories are prepended to loadPath.

[quietdec] when true, turns off the interactive system's prompt and responses, except warnings and error messages. Useful for writing scripts in SML. The default value is false; can be set to true with the -quietdec command line option.

[verbose] determines whether the signature inferred by a call to compile will be printed. The printed signature follows the syntax of Moscow ML signatures, so the output of compile "U.sml" can be edited to subsequently create file U.sig. The default value is ref false.

[quotation] determines whether quotations and antiquotations are permitted in declarations entered at top-level and in files compiled with compile. A quotation is a piece of text surrounded by backquote characters 'a b c' and is used to embed object language phrases in ML programs; see the Moscow ML Owner's Manual for a brief explanation of quotations. When quotation is false, the backquote character is an ordinary symbol which can be used in ML symbolic identifiers. When quotation is true, the backquote character is illegal in symbolic identifiers, and a quotation 'a b c' will be recognized by the parser and evaluated to an object of type 'a General.frag list. False by default.

[valuepoly] determines whether value polymorphism is used or not in the type checker. With value polymorphism (the default), there is no distinction between imperative ('_a) and applicative ('a) type variables, and type variables are generalized only in bindings to non-expansive expressions. Non-generalized type variables are left free, to be instantiated when the bound identifier is used. An expression is non-expansive if it is a variable, a special

72 META

constant, a function, a tuple or record of non-expansive expressions, a parenthesized or typed non-expansive expression, or the application of an exception or value constructor (other than ref) to a non-expansive expression. If valuepoly is false, then the type checker will distinguish imperative and applicative type variables, generalize all applicative type variables, and generalize imperative type variables only in non-expansive expressions. True by default.

MOSML 73

Module Mosml

Mosml -- some Moscow ML specific functions

datatype runresult =
 Success of string
 Failure of string

val run : string -> string list -> string -> runresult

[argv ()] returns the command line strings of the current process. Hence List.nth(argv (), 0) is the command used to invoke the SML process, List.nth(argv (), 1) is its first argument, and so on. We recommend using the SML Basis Library CommandLine structure instead.

[time f arg] applies f to arg and returns the result; as a side effect, it prints the time (cpu, system, and real time) consumed by the evaluation.

[listDir path] returns the list of all files and subdirectories of the directory indicated by path. Raises OS.SysErr in case of failure.

[doubleVec r] returns an eight-element vector of Word8.word, which contains the real number in the IEEE 754 floating-point 'double format' bit layout stored in big-endian (high byte first) order.

[vecDouble v] accepts an eight-element vector v of Word8.word, and returns the real number obtained by taking v to be an IEEE 754 floating-point 'double format' number stored in big-endian (high byte first) order. Raises Fail if v is not en eight-element vector.

[floatVec r] returns a four-element vector of Word8.word, which contains the real number in the IEEE 754 floating-point 'float format' bit layout stored in big-endian (high byte first) order. Raises Fail if r is not representable as a 32-bit float.

[vecFloat v] accepts a four-element vector v of Word8.word, and returns the real obtained by taking v to be an IEEE 754 floating-point 'float format' number stored in big-endian (high byte first) order. Raises Fail if v is not a four-element vector.

[md5sum s] computes the 128-bit MD5 checksum of string s and returns it as a 22 character base64 string.

[run cmd args inp] executes the program cmd with command-line arguments args and standard input inp. Returns Success s where s is the program's (standard and error) output as a string, if it executed successfully; otherwise returns Failure s where s is its (standard and error) output as a string.

74 MOSMLCGI

Module Mosmlcgi

```
Mosmlcgi -- support for writing CGI scripts in Moscow ML
```

1. Accessing the fields or parameters of a CGI call

2. Accessing parts in multipart/form-data; form-based file upload

val cgi_partnames : string list

type part

val part_field_strings : part -> string -> string list
val part_field_string : part -> string -> string option
val part_field_integer : part -> string * int -> int

3. Administrative information

```
val cgi_server_software
                                                                                              : string option
val cgi_server_name
val cgi_gateway_interface
                                                                                               : string option
                                                                                          : string option
                                                                                         : string option
val cgi_server_protocol
                                                                                              : string option
val cgi_server_port
                                                                                           : string option
val cgi_request_method
                                                                               string optionstring optionstring option
val cgi http accept
val cgi_http_user_agent
val cgi_http_referer
                                                                                              : string option
                                                                                           : string option
val cgi_path_info
val cgi_query_string
                                                                                             : string option
val cgi_remote_addr
val cgi_remote_user
val cgi_remote_ident
val cgi_auth_type
val cgi_auth_type
val cgi_auth_type
val cgi_auth_type
val cgi_auth_type

    string option
    st
val cgi_remote_host
                                                                                         : string option
string option
val cgi_content_length string option
val cgi_annotation_server string option
val cgi_annotation_server
val cgi_http_cookie
                                                                                              : string option
val cgi_http_host
                                                                                              : string option
val cgi_http_proxy_connection : string option
val cgi_script_filename : string option
val cgi_document_root
                                                                                             : string option
val cgi_server_admin
                                                                                              : string option
                                                                                            : string option
val cgi_api_version
val cgi_the_request
val cgi_request_uri
                                                                                           : string option

      val cgi_request_uri
      : string option

      val cgi_request_filename
      : string option

      val cgi_request_filename
      : string option
```

The Mosmlcgi library is for writing CGI programs in Moscow ML. A CGI program may be installed on a WWW server and is invoked in response to HTTP requests sent to the server from a web browser, typically from an HTML FORM element.

1. Obtaining field values sent from an ordinary HTML form

MOSMLCGI 75

[cgi_fieldnames] is a list of the names of fields present in the CGI call message. If field name fnm is in cgi_fieldnames, then cgi_field_string fnm <> NONE.

[cgi_field_strings fnm] is a (possibly empty) list of the strings bound to field fnm.

[cgi_field_string fnm] returns SOME(s) where s is a string bound to field name fnm, if any; otherwise NONE. Equivalent to case cgi_field_strings fnm of

```
[] => NONE
| s :: _ => SOME s
```

[cgi_field_integer (fnm, deflt)] attempts to parse an integer from field fnm. Returns i if cgi_field_string(fnm) = SOME(s) and an integer i can be parsed from a prefix of s; otherwise returns deflt.

2. Obtaining field values sent with ENCTYPE="multipart/form-data"

[cgi_partnames] is a list of the names of the parts of the multipart/form-data message.

The type part is the abstract type of parts of a message. Each part may have several fields. In this implementation, the field of a part cannot be a another part itself.

[cgi_parts pnm] is a (possibly empty) list of the parts called pnm.

[cgi_part pnm] is SOME(prt) where prt is a part called pnm, if any; otherwise NONE. Equivalent to

```
case cgi_parts pnm of
   [] => NONE
   | prt :: _ => SOME prt
```

[part_fieldnames prt] is the list of field names in part pnm.

[part_type prt] is SOME(typ) if the part prt contains a specification 'Context-Type: typ'; otherwise NONE.

[part_data prt] is the data contain in part prt; for instance, the contents of a file uploaded via form-based file upload.

[part_field_strings prt fnm] is a (possibly empty) list of the strings bound to field fnm in part prt.

[part_field_string prt fnm] returns SOME(s) where s is a string bound to field name fnm in part prt, if any; otherwise NONE. Equivalent to

```
case part_field_strings prt fnm of
   [] => NONE
   | s :: _ => SOME s
```

[part_field_integer prt (fnm, deflt)] attempts to parse an integer from field fnm of part prt. Returns i if part_field_string prt fnm = SOME(s) and an integer i can be parsed from a prefix of s; otherwise returns deflt.

3. Administrative and server information

Each of the following variables has the value SOME(s) if the corresponding CGI environment variable is bound to string s; otherwise NONE:

```
[cgi_server_software] is the value of SERVER_SOFTWARE
```

[cgi_server_name] is the value of SERVER_NAME

76 MOSMLCGI

```
[cgi_gateway_interface] is the value of GATEWAY_INTERFACE
[cqi server protocol] is the value of SERVER PROTOCOL
[cgi_server_port] is the value of SERVER_PORT
[cgi_request_method] is the value of REQUEST_METHOD
[cgi http accept] is the value of HTTP ACCEPT
[cgi_http_user_agent] is the value of HTTP_USER_AGENT
[cgi_http_referer] is the value of HTTP_REFERER
[cgi_path_info] is the value of PATH_INFO
[cgi_path_translated] is the value of PATH_TRANSLATED
[cgi_script_name] is the value of SCRIPT_NAME
[cgi_query_string] is the value of QUERY_STRING
[cgi_remote_host] is the value of REMOTE_HOST
[cgi_remote_addr] is the value of REMOTE_ADDR
[cgi_remote_user] is the value of REMOTE_USER
[cgi_remote_ident] is the value of REMOTE_IDENT
[cgi_auth_type] is the value of AUTH_TYPE
[cgi_content_type] is the value of CONTENT_TYPE
[cgi_content_length] is the value of CONTENT_LENGTH, that is, the
length of the data transmitted in the CGI call.
[cgi\_annotation\_server] is the value of ANNOTATION_SERVER
[cgi_http_cookie] is the value of HTTP_COOKIE
[cgi_http_forwarded] is the value of HTTP_FORWARDED
[cgi_http_host] is the value of HTTP_HOST
[cgi_http_proxy_connection] is the value of HTTP_PROXY_CONNECTION
[cgi_script_filename] is the value of SCRIPT_FILENAME
[cgi_document_root] is the value of DOCUMENT_ROOT
[cgi_server_admin] is the value of SERVER_ADMIN
[cgi_api_version] is the value of API_VERSION
[cgi_the_request] is the value of THE_REQUEST
[cgi_request_uri] is the value of REQUEST_URI
[cgi_request_filename] is the value of REQUEST_FILENAME
[cgi_is_subreq] is the value of IS_SUBREQ
```

MOSMLCOOKIE 77

Module Mosmlcookie

```
Mosmlcookie -- getting and setting cookies in CGI scripts
exception CookieError of string
                   : string list
val allCookies
val getCookieValue : string -> string option
val getCookie
                  : string -> string option
type cookiedata =
    { name : string, value : string,
      expiry : Date.date option,
      domain : string option,
      path : string option,
      secure : bool }
val setCookie
                 : cookiedata -> string
val setCookies : cookiedata -> string val setCookies : cookiedata list -> string
val deleteCookie : { name : string, path : string option } -> string
```

These functions may be used in CGI scripts to get and set cookies. (c) Hans Molin, Computing Science Dept., Uppsala University, 1999.

[getCookieValue ck] returns SOME(v) where v is the value associated with the cookie ck, if any; otherwise returns NONE.

[getCookie ck] returns SOME(nv) where nv is the ck=value string for the cookie ck, if any; otherwise returns NONE.

[allCookies] is a list [nv1, nv2, ..., nvm] of all the ck=value pairs of defined cookies.

[setCookie { name, value, expiry, domain, path, secure }] returns a string which (when transmitted to a browser as part of the HTTP response header) sets a cookie with the given name, value, expiry date, domain, path, and security.

[setCookies ckds] returns a string which (when transmitted to a browser as part of the HTTP response header) sets the specified cookies.

[deleteCookie { name, path }] returns a string which (when transmitted to a browser as part of the HTTP response header) deletes the specified cookie by setting its expiry to some time in the past.

78 MSP

Module Msp

```
Msp -- utilities for CGI scripts and ML Server Pages
Efficiently concatenable word sequences
datatype wseq =
    Empty
                                       The empty sequence
   Νl
                                       Newline
    $ of string
                                       A string
                                       A sequence of strings
    $$ of string list
  && of wseq * wseq;
                                       Concatenation of sequences
Manipulating wseqs
val prmap
            : ('a -> wseq) -> 'a list -> wseq
            : wseq -> ('a -> wseq) -> 'a list -> wseq
val prsep
val flatten : wseq -> string
val printseq : wseq -> unit
val vec2list : 'a vector -> 'a list
Shorthands for accessing CGI parameters
exception ParamMissing of string
exception NotInt of string * string
            : string -> string
val %
val %?
            : string -> bool
val %#
            : string -> int
            : string * string -> string
val %%
            : string * int -> int
val %%#
HTML generic marks
val mark0
            : string -> wseq
val mark0a
            : string -> string -> wseq
            : string -> wseq -> wseq
val mark1
val markla
            : string -> string -> wseq -> wseq
val comment : wseq -> wseq
HTML documents and headers
val html
            : wseq -> wseq
val head
            : wseq -> wseq
val title
            : wseq -> wseq
val body
            : wseq -> wseq
            : string -> wseq -> wseq
val bodya
val htmldoc : wseq -> wseq -> wseq
HTML headings and vertical format
val h1
            : wseq -> wseq
val h2
            : wseq -> wseq
            : wseq -> wseq
val h3
val h4
            : wsea -> wsea
            : wseq -> wseq
val h5
            : wseq -> wseq
val h6
val p
            : wseq -> wseq
val pa
            : string -> wseq -> wseq
            : wseq
val br
val bra
            : string -> wseq
           : wseq
val hr
val hra
            : string -> wseq
               : wseq -> wseq
val divi
val divia
               : string -> wseq -> wseq
val blockquote : wseq -> wseq
val blockquotea : string -> wseq -> wseq
              : wseq -> wseq
val center
```

MSP 79

```
val address
               : wseq -> wseq
                : wseq -> wseq
val pre
HTML anchors and hyperlinks
val ahref
             : string -> wseq -> wseq
val ahrefa : string -> string -> wseq -> wseq
           : string -> wseq -> wseq
val aname
HTML text formats and style
: wseq -> wseq
val tt
            : wseq -> wseq
: wseq -> wseq
val sub
val sup
val fonta : string -> wseq -> wseq
HTML lists
val ul
            : wseq -> wseq
            : string -> wseq -> wseq
: wseq -> wseq
val ula
val ol
val ola
            : string -> wseq -> wseq
val li
            : wseq -> wseq
val dl
             : wseq -> wseq
            : string -> wseq -> wseq
val dla
val dt
            : wseq -> wseq
val dd
            : wseq -> wseq
HTML tables
val table
             : wseq -> wseq
val tablea : string -> wseq -> wseq
val tr
            : wseq -> wseq
             : string -> wseq -> wseq
val tra
            : wseq -> wseq
val td
            : string -> wseq -> wseq
val tda
            : wseq -> wseq
val th
val tha
             : string -> wseq -> wseq
val caption : wseq -> wseq
val captiona : string -> wseq -> wseq
HTML images and image maps
             : string -> wseq
val img
val imga
             : string -> string -> wseq
val map
             : string -> wseq -> wseq
val mapa
             : string -> string -> wseq -> wseq
             : { alt : string option, coords : string, href : string option, shape : string} -> wseq
val area
HTML forms etc
               : string -> wseq -> wseq
val form
val forma
              : string -> string -> wseq -> wseq
               : string -> wseq
val input
             : string -> string -> wseq
: string -> string -> wseq
val inputa
val inpassword : string > string > wseq
val incheckbox : {name : string, value : string} -> string -> wseq
val inradio : {name : string, value : string} -> string -> wseq
val textarea : string -> wseq -> wseq
val textareaa : string -> string -> wseq -> wseq
               : string -> string -> wseg -> wseg
val select
val option
               : string -> wseq
```

HTML frames and framesets

```
val frameset : string -> wseq -> wseq
val frame : { src : string, name : string } -> wseq
val framea : { src : string, name : string } -> string -> wseq

HTML encoding
val urlencode : string -> string
val htmlencode : string -> string
```

This module provides support functions for writing CGI scripts and ML Server Page scripts.

[wseq] is the type of efficiently concatenable word sequences. Building an HTML page (functionally) as a wseq is more efficient than building it (functionally) as a string, and more convenient and modular than building it (imperatively) by calling print.

[Empty] represents the empty string "".

[Nl] represents the string "\n" consisting of a single newline character.

[\$ s] represents the string s.

[\$\$ ss] represents the string String.concat(ss).

[&&(ws1, ws2)] represents the concatenation of the strings represented by ws1 and ws2. The function && should be declared infix &&

[prmap f xs] is f x1 && ... && f xn evaluated from left to right, when xs is [x1, ..., xn].

[prsep sep f xs] is f x1 && sep && ... && sep && f xn, evaluated from left to right, when xs is [x1, ..., xn].

[flatten ws] is the string represented by ws.

[printseq ws] is equivalent to print(flatten ws), but avoids building any new strings.

[vec2list vec] is a list of the elements of vector vec. Use it to convert e.g. the results of a database query into a list, for processing with prmap or prsep.

Shorthands for accessing CGI parameters:

[%? fnm] returns true if there is a string associated with CGI parameter fnm; otherwise returns false.

[% fnm] returns a string associated with CGI parameter fnm if there is any; raises ParamMissing(fnm) if no strings are associated with fnm. Equivalent to

case Mosmlcgi.cgi_field_string fnm of
 NONE => raise ParamMissing "fnm"
 | SOME v => v

In general, multiple strings may be associated with a CGI parameter; use Mosmlcgi.cgi_field_strings if you need to access all of them.

[%# fnm] returns the integer i if there is a string associated with CGI parameter fnm, and that string is parsable as ML integer i. Raises ParamMissing(fnm) if no string is associated with fnm. Raises NotInt(fnm, s) if there is a string but it is not parsable as an ML int.

[%%(fnm, dflt)] returns a string associated with CGI parameter fnm if there is any; otherwise returns the string dflt.

[%%#(fnm, dflt)] returns the integer i if there is a string associated with CGI parameter fnm, and that string is parsable as

an ML int; otherwise returns the string dflt. HTML generic marks: [mark0 t] generates the HTML tag <t> as a wseq. [mark0a attr t] generates the attributed HTML tag <t attr> as a wseq. [mark1 t ws] generates <t>ws</t> as a wseq. [markla attr t ws] generates <t attr>ws</t> as a wseq. [comment ws] generates <!--ws--> as a wseq. HTML documents and headers: [html ws] generates <hTML>ws</hTML>. [head ws] generates <HEAD>ws</HEAD>. [title ws] generates <TITLE>ws</TITLE>. [body ws] generates <BODY>ws</BODY>. [bodya attr ws] generates <BODY attr>ws</BODY>. [htmldoc titl ws] generates <HTML><HEAD><TITLE>titl</TITLE></HEAD><BODY>ws</BODY></HTML>. HTML headings and vertical format: [h1 ws] generates <H1>ws</H1>. [p ws] generates <P>ws</P>. [pa attr ws] generates <P attr>ws</P>. [br] generates
. [bra attr] generates <BR attr>. [hr] generates <HR>. [hra attr] generates <HR attr>. [divi ws] generates <DIV>ws</DIV>. [divia attr ws] generates <DIV attr>ws</DIV>. [blockquote ws] generates <BLOCKQUOTE>ws</BLOCKQUOTE>. [blockquotea attr ws] generates <BLOCKQUOTE attr>ws</BLOCKQUOTE> [center ws] generates <CENTER>ws</CENTER>. [address ws] generates <ADDRESS>ws</ADDRESS>. [pre ws] generates <PRE>ws</PRE>. HTML anchors and hyperlinks: [ahref link ws] generates ws. [ahrefa link attr ws] generates ws. [aname nam ws] generates ws.

```
HTML text formats and style:
[em ws] generates <EM>ws</EM>.
[strong ws] generates <STRONG>ws</STRONG>.
[tt ws] generates <TT>ws</TT>.
[sub ws] generates <SUB>ws</SUB>.
[sup ws] generates <SUP>ws</SUP>.
[fonta attr ws] generates <FONT attr>ws</FONT>.
HTML lists:
[ul ws] generates <UL>ws</UL>.
[ula attr ws] generates <UL attr>ws</UL>.
[ol ws] generates <OL>ws</OL>.
[ola attr ws] generates <OL attr>ws</OL>.
[li ws] generates <LI>ws</LI>.
[dl ws] generates <DL>ws</DL>.
[dla attr ws] generates <DL attr>ws</DL>.
[dt ws] generates <DT>ws</DT>.
[dd ws] generates <DD>ws</DD>.
HTML tables:
[table ws] generates <TABLE>ws</TABLE>.
[tablea attr ws] generates <TABLE attr>ws</TABLE>.
[tr ws] generates <TR>ws</TR>.
[tra attr ws] generates <TR attr>ws</TR>.
[td ws] generates <TD>ws</TD>.
[tda attr ws] generates <TD attr>ws</TD>.
[th ws] generates <TH>ws</TH>.
[tha attr ws] generates <TH attr>ws</TH>.
[caption ws] generates <CAPTION>ws</CAPTION>.
[captiona attr ws] generates <CAPTION attr>ws</CAPTION>.
HTML images and image maps:
[img s] generates <IMG SRC="s">.
[imga s attr] generates <IMG SRC="s" attr>.
[map nam ws] generates <MAP NAME="name">ws</MAP>.
[mapa nam attr ws] generates <MAP NAME="name" attr>ws</MAP>.
[area { alt, coords, href, shape}] generates
<AREA SHAPE="shape" COORDS="coords" HREF="link" ALT="desc">
when href is SOME link (where HREF is replaced by NOHREF otherwise)
```

and alt is SOME desc (where ALT is omitted otherwise). HTML forms etc: [form act ws] generates <FORM ACTION="act">ws</FORM>. [forma act attr ws] generates <FORM ACTION="act" attr>ws</FORM>. [input typ] generates <INPUT TYPE=typ>. [inputa typ attr] generates <INPUT TYPE=typ attr>. [intext name attr] generates <INPUT TYPE=TEXT NAME="name" attr>. [inpassword name attr] generates <INPUT TYPE=PASSWORD NAME="name" attr>. [incheckbox {name, value} attr] generates <INPUT TYPE=CHECKBOX NAME="name" VALUE="value" attr>. [inradio {name, value} attr] generates <INPUT TYPE=RADIO NAME="name" VALUE="value" attr>. [inreset value attr] generates <INPUT TYPE=RESET VALUE="value" attr>. [insubmit value attr] generates <INPUT TYPE=SUBMIT VALUE="value" attr>. [inhidden {name, value}] generates <INPUT TYPE=HIDDEN NAME="name" VALUE="value">. [textarea name ws] generates <TEXTAREA NAME="name">ws</TEXTAREA>. [textareaa name attr ws] generates <TEXTAREA NAME="name" attr>ws</TEXTAREA>. [select name attr ws] generates <SELECT NAME="name" attr>ws</SELECT>. [option value] generates <OPTION VALUE="value">. HTML frames and framesets: [frameset attr ws] generates <FRAMESET attr>ws</FRAMESET>. [frame { src, name }] generates <FRAME SRC="src" NAME="name">. [framea { src, name } attr] generates <FRAME SRC="src" NAME="name" attr>. HTML encoding functions: [urlencode s] returns the url-encoding of s. That is, space (ASCII 32) is replaced by '+' and every non-alphanumeric character c except the three characters hyphen (-), underscore (_) and full stop (.) is replaced by %hh, where hh is the hexadecimal representation of the ASCII code of c. [htmlencode s] returns the html-encoding of s. That is, < and > are replaced by < and > respectively, and & is replaced by &

84 MYSQL

Module Mysql

```
Mysql -- interface to the MySQL database server -- requires Dynlib
                                            Connection to server
type dbconn
                                             Result of a query
type dbresult
eqtype oid
                                             (not used by Mysql)
exception Closed
                                             Connection is closed
exception Null
                                            Field value is NULL
Opening, closing, and maintaining database connections
val openbase : { dbhost
                          : string option,
                                            database server host
                          : string option,
                dbname
                                            database name
                dboptions: string option,
                                            (not used by Mysql)
                        : string option,
                                            database server port
                dbport
                                            user passwd (not used by Mysql)
                dbpwd
                          : string option,
                dbtty
                         : string option,
                dbuser
                        : string option
                                            database user
              } -> dbconn
val closebase
              : dbconn -> unit
val db
               : dbconn -> string
val host
                : dbconn -> string option
               : dbconn -> string
val options
               : dbconn -> string
val port
                : dbconn -> string
val tty
                : dbconn -> bool
val status
                : dbconn -> unit
val reset
val errormessage : dbconn -> string option
Query execution and result set information
datatype dbresultstatus =
    Bad_response
                           (not used by Mysql)
    Command ok
                           The query was a command
                          (not used by Mysql)
    Copy in
                          (not used by Mysql)
   Copy_out
   Empty_query
    Fatal_error
                           (not used by Mysql)
   Nonfatal_error
                           The query successfully returned tuples
  Tuples_ok
val execute
              : dbconn -> string -> dbresult
val resultstatus : dbresult -> dbresultstatus
                : dbresult -> int
val ntuples
val cmdtuples
                : dbresult -> int
               : dbresult -> int
val nfields
val fname
                : dbresult -> int -> string
val fnames
                : dbresult -> string vector
               : dbresult -> string -> int option
val fnumber
Accessing the fields of a resultset
                : dbresult -> int -> int -> int
val getint
                : dbresult -> int -> int -> real
val getreal
               dbresult -> int -> int -> string
dbresult -> int -> int -> int * int *
dbresult -> int -> int -> int * int *
dbresult -> int -> int -> int * int *
val getstring
val getdate
val gettime
val getdatetime : dbresult -> int -> int -> Date.date
datatype dynval =
    Bool of bool
                                       (not used by Mysql)
   Int of int
                                      Mysql int4
   Real of real
                                      Mysql float8 (float4)
  String of string
                                      Mysql text (varchar)
```

```
Date of int * int * int
                                     Mysql date yyyy-mm-dd
   Time of int * int * int
                                      Mysql time hh:mm:ss
   DateTime of Date.date
                                      Mysql datetime
   Oid of oid
                                       (not used by Mysql)
   Bytea of Word8Array.array
                                       (not used by Mysql)
   NullVal
                                       Mysql NULL value
val getdynfield : dbresult -> int -> int -> dynval
               : dbresult -> int -> dynval vector
: dbresult -> dynval vector vector
: dynval -> string
val getdyntup
val getdyntups
val dynval2s
Bulk copying to or from a table
val copytableto : dbconn * string * (string -> unit) -> unit
val copytablefrom : dbconn * string * ((string -> unit) -> unit) -> unit
Some standard ML and MySQL types:
datatype dyntype =
                       ML bool
    BoolTy
                                               (not used by Mysql)
                      ML int
                                               Mysql int4
   IntTy
   RealTy
                      ML real
                                               Mysql float8, float4
   StringTy
                     ML string
                                               Mysql text, varchar
                     ML (yyyy, mth, day) Mysql date
   DateTy
                     ML (hh, mm, ss)
   TimeTy
                                               Mysql time
   DateTimeTy
                     ML Date.date
                                               Mysql datetime, abstime
                     {\tt ML} oid
   OidTy
                                               (not used by Mysql)
   ByteArrTy
                      ML Word8Array.array (not used by Mysql)
  UnknownTy of oid
val fromtag : dyntype -> string
val ftype : dbresult -> int -> dyntype
val ftypes : dbresult -> dyntype Vector.vector
val applyto : 'a -> ('a -> 'b) -> 'b
Formatting the result of a database query as an HTML table
val formattable : dbresult -> Msp.wseq
val showquery : dbconn -> string -> Msp.wseq
```

[dbconn] is the type of connections to a MySQL database.

[dbresult] is the type of result sets from MySQL queries.

[openbase { dbhost, dbport, dboptions, dbtty, dbname, dbuser, dbpwd }] opens a connection to a MySQL database server on the given host (default the local one) on the given port (default?), to the given database (defaults to the user's login name), for the given user name (defaults to the current user's login name), and the given password (default none). The result is a connection which may be used in subsequent queries. In MySQL, unlike PostgreSQL, the dboptions and dbtty fields are not used.

[closebase dbconn] closes the database connection. No further queries can be executed.

[db dbconn] returns the name of the database.

[host dbconn] returns SOME h, where h is the database server host name, if the connection uses the Internet; returns NONE if the connection is to a socket on the local server.

[options dbconn] returns the options given when opening the database.

[port dbconn] returns the port number of the connection.

[tty dbconn] returns the name of the tty used for logging.

86 MYSQL

[status dbconn] returns true if the connection is usable, false otherwise.

[reset dbconn] attempts to close and then reopen the connection to the database server.

[errormessage dbconn] returns NONE if no error occurred, and SOME msg if an error occurred, where msg describes the error.

[execute dbconn query] sends an SQL query to the database server for execution, and returns a resultset dbres.

[resultstatus dbres] returns the status of the result set dbres. After a select query that succeeded, it will be Tuples_ok.

[ntuples dbres] returns the number of tuples in the result set after a query.

[cmdtuples dbres] returns the number of tuples affected by an insert, update, or delete SQL command.

[nfields dbres] returns the number of fields in each tuple after a query.

[fname dbres fno] returns the name of field number fno (in the result set after a query). The fields are numbered 0, 1,...

[fnames dbres] returns a vector of the field names (in the result set after a query).

[fnumber dbres fname] returns SOME i where i is the number (0, 1, ...) of the field called fname (in the result set after a query), if the result set contains such a field name; returns NONE otherwise.

[ftype dbres fno] returns the dyntype of field number fno (in the result set after a query).

[ftypes dbres] returns a vector of the dyntypes (in the result set after a query).

[fromtag dt] returns the name of the preferred MySQL type used to represent values of the dyntype dt. This may be used when building 'create table' statements.

[getint dbres fno tupno] returns the integer value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[getreal dbres fno tupno] returns the floating-point value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[getstring dbres fno tupno] returns the string value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[getdate dbres fno tupno] returns the date (yyyy, mth, day) value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL. Raises Fail if the field cannot be scanned as a date.

[gettime dbres fno tupno] returns the time-of-day (hh, mm, ss) value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL. Raises Fail if the field cannot be scanned as a time.

[getdatetime dbres fno tupno] returns the Date.date value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL. Raises Fail if the field cannot be scanned as a date.

[getbool dbres fno tupno] returns the boolean value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[isnull dbres fno tupno] returns true if the value of field number fno in tuple tupno of result set dbres is NULL; false otherwise.

[getdynfield dbres fno tupno] returns the value of field number fno in tuple tupno of result set dbres as a dynval (a wrapped value). A NULL value is returned as NullVal. Note that the partial application (getdynfield dbres fno) precomputes the type of the field fno. Hence it is far more efficient to compute

let val getfno = getdynfield dbres fno
in tabulate(ntuples dbres, getfno) end

than to compute

let fun getfno tupno = getdynfield dbres fno tupno
in tabulate(ntuples dbres, getfno) end
because the latter repeatedly computes the type of the field.

[getdyntup dbres tupno] returns the fields of tuple tupno in result

[getdyntups dbres] returns all tuples of result set dbres as a vector of vectors of dynvals.

[dynval2s dv] returns a string representing the dynval dv.

[applyto x f] computes f(x). This is convenient for applying
several functions (given in a list or vector) to the same value:
 map (applyto 5) (tabulate(3, getdynfield dbres))
equals

[getdynfield dbres 0 5, getdynfield dbres 1 5, getdynfield dbres 2 5]

[copytableto(dbconn, tablename, put)] simulates a PostgreSQL "COPY
TABLE TO" statement, applies the function put to every tuple of the
table, represented as a line of text (not terminated by newline
\n), and cleans up at the end. For instance, to copy the contents
of a table t to a text stream s (one tuple on each line), define
 fun put line =

 $\begin{tabular}{ll} \hline (TextIO.output(s, line); TextIO.output(s, "\n")) \\ and execute \\ \hline \end{tabular}$

copytableto(dbconn, "t", put).

set dbres as a vector of dynvals.

[copytablefrom(dbconn, tablename, useput)] simulates a PostgreSQL "COPY TABLE FROM" statement, creates a put function for copying lines to the table, passes the put function to useput, and cleans up at the end. The put function may be called multiple times for each line (tuple); the end of each line is indicated by the newline character "\n" as usual. For instance, to copy the contents of a text stream s to a table t, define

fun useput put =
 while not (TextIO.endOfStream s) do put(TextIO.inputLine s);

and execute
 copytablefrom(dbconn, "t", useput).

Note that TextIO.inputLine preserves the newline at the end of each line.

[formattable dbresult] returns a wseq representing an HTML table. The HTML table has a column for every field in the dbresult. The first row is a table header giving the names of the fields in the dbresult. The remaining rows correspond to the tuples in the dbresult, in the order they are provided by the database server. Null fields are shown as NULL.

[showquery dbconn query] sends the SQL query to the database server, then uses formattable to format the result of the query.

88 NJ93

Module NJ93

```
NJ93 -- compatibility SML/NJ 0.93 top-level environment
             : string -> unit
val print
NJ93 Integer
             : int * int -> int : int * int -> int
val max
val min
NJ93 List
exception Hd and Tl and Nth and NthTail
               : 'a list -> 'a
: 'a list -> 'a list
val hd
                                             Hd
val tl
               : 'a list * int -> 'a
val nth
                                             Nth
             : 'a list * int -> 'a list NthTail
val nthtail
               : ('a -> 'b) -> 'a list -> unit
val app
val revapp : ('a -> 'b) -> 'a list -> unit val fold : ('a * 'b -> 'b) -> 'a list -> 'b -> 'b val revfold : ('a * 'b -> 'b) -> 'a list -> 'b -> 'b
NJ93 Real
               : real -> int
val ceiling
val truncate : real -> int
NJ93 Ref
val inc
             : int ref -> unit
val dec
               : int ref -> unit
NJ93 String
exception Substring
               : string * int -> int
val ordof
val ord
              : string -> int
                                                   Ord
              : int -> string
val chr
                                                   Chr
val substring : string * int * int -> string
                                                   Substring
             : string -> string list : string list -> string
val explode
val implode
NJ93 top-level math functions
               : real -> real
val sqrt
val sin
              : real -> real
              : real -> real
val cos
              : real -> real
val arctan
val exp
              : real -> real
               : real -> real
val ln
NJ93 top-level input/output, standard
type instream and outstream
val std in
                   : instream
val open_in
                   : string -> instream
                   : instream * int -> string
val input
val lookahead
                  : instream -> string
                 : instream -> unit
val close_in
val end_of_stream : instream -> bool
val std_out
                  : outstream
                   : string -> outstream
val open_out
val output
                   : outstream * string -> unit
val close_out
                   : outstream -> unit
NJ93 top-level input/output, non-standard
```

NJ93 89

val open_in_bin : string -> instream
val open_out_bin : string -> outstream
val inputc : instream -> int -> string
val std_err : outstream

val std_err : Outstream
val outputc : outstream -> string -> unit
val flush_out : outstream -> unit
val input_line : instream -> string
val can_input : instream * int -> bool
val open_append : string -> outstream

90 NONSTDIO

Module Nonstdio

Nonstdio -- non-standard I/O -- use BinIO and TextIO instead local open BasicIO in : string -> instream val open_in_bin val buff_input : instream -> CharArray.array -> int -> int -> int val input_value : instream -> 'a : instream -> int -> unit val seek_in val pos_in : instream -> int val in_stream_length : instream -> int val fast_really_input : instream -> string -> int -> int -> unit val open_out_bin
val open_out_exe
val output_char
: string -> outstream
: string -> outstream
: outstream
: outstream : outstream -> Char.char -> unit val output_char val output_char
val output_byte
val buff_output
val buff_output
val outstream -> int -> unit
val buff_output
: outstream -> CharArray.array -> int -> int -> unit val output_binary_int : outstream -> int -> unit val output_value : outstream -> 'a -> unit
val seek_out : outstream -> int -> unit val seek_out val pos_out : outstream -> int val file_exists : string -> bool

end

OS 91

Module OS

end

OS -- SML Basis Library

signature OS = sig

type syserror = syserror

exception SysErr of string * syserror option

val errorMsg : syserror -> string

structure FileSys : FileSys
structure Path : Path
structure Process : Process

[errorMsg err] returns a string explaining the error message system error code err, as found in a SysErr exception. The precise form of the string depends on the operating system.

92 OPTION

Module Option

Option -- SML Basis Library

```
exception Option
datatype option = datatype option
val getOpt
                        : 'a option * 'a -> 'a
                        : 'a option -> bool
val isSome
                        : 'a option -> 'a
val valOf
                        : ('a -> bool) -> 'a -> 'a option
val filter
                        : ('a -> 'b) -> 'a option -> 'b option
: ('a -> unit) -> 'a option -> unit
val map
val app
                        : 'a option option -> 'a option
val join
val compose : ('a -> 'b) * ('c -> 'a option) -> ('c -> 'b option)
val mapPartial : ('a -> 'b option) -> ('a option -> 'b option)
val composePartial : ('a -> 'b option) * ('c -> 'a option) -> ('c -> 'b option)
    [getOpt (xopt, d)] returns x if xopt is SOME x; returns d otherwise.
    [isSome vopt] returns true if xopt is SOME x; returns false otherwise.
    [valOf vopt] returns x if xopt is SOME x; raises Option otherwise.
```

[filter p x] returns SOME x if p x is true; returns NONE otherwise.

[map f xopt] returns SOME (f x) if xopt is SOME x; returns NONE otherwise.

[mapPartial f xopt] returns f x if xopt is SOME x; returns NONE otherwise.

[app f xopt] applies f to x if xopt is SOME x; does nothing otherwise.

[compose (f, g) x] returns SOME (f y) if g x is SOME y; returns NONE

[composePartial $(f, g) \times]$ returns f y if g x is SOME y; returns NONE otherwise. It holds that composePartial (f, g) = mapPartial f o g.

[join xopt] returns x if xopt is SOME x; returns NONE otherwise.

otherwise. It holds that compose (f, g) = map f o g.

It holds that mapPartial f = join o map f.

The operators (map, join, SOME) form a monad.

PP 93

Module PP

```
PP -- pretty-printing -- from the SML/NJ library
type ppconsumer = { consumer : string -> unit,
                   linewidth : int,
                            : unit -> unit }
                   flush
datatype break_style =
   CONSISTENT
  INCONSISTENT
val mk_ppstream
                   : ppconsumer -> ppstream
val dest_ppstream : ppstream -> ppconsumer
val add_break
                  : ppstream -> int * int -> unit
val add_newline
                  : ppstream -> unit
                  : ppstream -> string -> unit
val add_string
                  : ppstream -> break_style -> int -> unit
val begin block
val end_block
                  : ppstream -> unit
val clear_ppstream : ppstream -> unit
val flush_ppstream : ppstream -> unit
val with_pp
                  : ppconsumer -> (ppstream -> unit) -> unit
val pp_to_string : int -> (ppstream -> 'a -> unit) -> 'a -> string
```

This structure provides tools for creating customized Oppen-style pretty-printers, based on the type ppstream. A ppstream is an output stream that contains prettyprinting commands. The commands are placed in the stream by various function calls listed below.

There following primitives add commands to the stream: begin_block, end_block, add_string, add_break, and add_newline. All calls to add_string, add_break, and add_newline must happen between a pair of calls to begin_block and end_block must be properly nested dynamically. All calls to begin_block and end_block must be properly nested (dynamically).

[ppconsumer] is the type of sinks for pretty-printing. A value of type ppconsumer is a record

{ consumer : string -> unit, linewidth : int, flush : unit -> unit }

of a string consumer, a specified linewidth, and a flush function which is called whenever flush_ppstream is called.

A prettyprinter can be called outright to print a value. In addition, a prettyprinter for a base type or nullary datatype ty can be installed in the top-level system. Then the installed prettyprinter will be invoked automatically whenever a value of type ty is to be printed.

[break_style] is the type of line break styles for blocks:

[CONSISTENT] specifies that if any line break occurs inside the block, then all indicated line breaks occur.

[INCONSISTENT] specifies that breaks will be inserted to only to avoid overfull lines.

[mk_ppstream {consumer, linewidth, flush}] creates a new ppstream which invokes the consumer to output text, putting at most linewidth characters on each line.

 $[{\tt dest_ppstream\ ppstrm}]$ extracts the linewidth, flush function, and consumer from a ppstream.

[add_break ppstrm (size, offset)] notifies the pretty-printer that a line break is possible at this point.

- * When the current block style is CONSISTENT:
 - ** if the entire block fits on the remainder of the line, then output size spaces; else
 - ** increase the current indentation by the block offset;

94 PP

further indent every item of the block by offset, and add one newline at every add_break in the block.

* When the current block style is INCONSISTENT:

- ** if the next component of the block fits on the remainder of the line, then output size spaces; else
- ** issue a newline and indent to the current indentation level plus the block offset plus the offset.

[add newline ppstrm] issues a newline.

[add_string ppstrm str] outputs the string str to the ppstream.

[begin_block ppstrm style blockoffset] begins a new block and level of indentation, with the given style and block offset.

[end_block ppstrm] closes the current block.

[clear_ppstream ppstrm] restarts the stream, without affecting the underlying consumer.

[flush_ppstream ppstrm] executes any remaining commands in the ppstream (that is, flushes currently accumulated output to the consumer associated with ppstrm); executes the flush function associated with the consumer; and calls clear_ppstream.

[with_pp consumer f] makes a new ppstream from the consumer and applies f (which can be thought of as a producer) to that ppstream, then flushed the ppstream and returns the value of f.

[pp_to_string linewidth printit x] constructs a new ppstream ppstrm whose consumer accumulates the output in a string s. Then evaluates (printit ppstrm x) and finally returns the string s.

Example 1: A simple prettyprinter for Booleans:

```
load "PP";
fun ppbool pps d =
   let open PP
   in
        begin_block pps INCONSISTENT 6;
        add_string pps (if d then "right" else "wrong");
        end_block pps
end;
```

Now one may define a ppstream to print to, and exercise it:

fun ppb b = (ppbool ppstrm b; PP.flush_ppstream ppstrm);

```
- ppb false;
wrong> val it = () : unit
```

The prettyprinter may also be installed in the toplevel system; then it will be used to print all expressions of type bool subsequently computed:

```
- installPP ppbool;
> val it = () : unit
- 1=0;
> val it = wrong : bool
- 1=1;
> val it = right : bool
```

See library Meta for a description of installPP.

PP 95

```
Example 2: Prettyprinting simple expressions (examples/pretty/ppexpr.sml):
    datatype expr =
        Cst of int
        Neg of expr
       Plus of expr * expr
    fun ppexpr pps e0 =
        let open PP
            fun ppe (Cst i)
                                    = add_string pps (Int.toString i)
                                 = (add_string pps "~"; ppe e)
               | ppe (Neg e)
               ppe (Plus(e1, e2)) = (begin_block pps CONSISTENT 0;
                                        add_string pps "(";
                                        ppe el;
                                        add_string pps " + ";
                                        add_break pps (0, 1);
                                        ppe e2;
add_string pps ")";
                                        end_block pps)
        in
            begin_block pps INCONSISTENT 0;
            ppe e\overline{0};
            end_block pps
    val _ = installPP ppexpr;
    Some example values:
    val e1 = Cst 1;
    val e2 = Cst 2;
    val e3 = Plus(e1, Neg e2);
    val e4 = Plus(Neg e3, e3);
    val e5 = Plus(Neg e4, e4);
val e6 = Plus(e5, e5);
    val e7 = Plus(e6, e6);
    val e8 =
        Plus(e3, Plus(e3, Plus(e3, Plus(e3, Plus(e3, Plus(e3, e7))))));
```

96 PARSING

Module Parsing

Parsing -- runtime library for parsers generated by mosmlyac Based on the runtime library for camlyacc; copyright 1993 INRIA, France local open Vector Obj Lexing in val symbolStart : unit -> int val symbolEnd : unit -> int : int -> int : int -> int val itemStart val itemEnd val clearParser : unit -> unit For internal use in generated parsers: type parseTables = (unit -> obj) vector * actions transl int vector * 1hs string * string * 1en defred string * string * dgoto sindex string * string * rindex gindex string tablesize int * table string * check string exception yyexit of obj exception ParseError of (obj -> bool) val yyparse : parseTables -> int -> (lexbuf -> 'a) -> lexbuf -> 'b

These functions are for use in mosmlyac-generated parsers. For further information, see the Moscow ML Owner's Manual. For examples, see mosml/examples/lexyacc and mosml/examples/calc.

A grammar definition (input to mosmlyac) consists of fragments of this form

val peekVal : int -> 'a

end

where the grsyms are sequences of grammar symbols, matching some string of characters, and the actions are corresponding semantic actions, written in ML. The following functions can be used in the semantic actions:

[symbolStart ()] returns the start position of the string that matches the sequence of grammar symbols. The first character in the input stream has position 0. May be called in a semantic action only.

[symbolEnd ()] returns the end position, plus one, of the string that matches the sequence of grammar symbols. The first character in the input stream has position 0. May be called in a semantic action only.

[itemStart i] returns the start position of the string that matches the i'th grammar symbol in the sequence. The first grammar symbol has number 1. The first character in the input stream has position 0. May be called in a semantic action only.

PARSING 97

[itemEnd i] returns the end position, plus one, of the string that matches the i'th grammar symbol in the sequence. The first grammar symbols has number 1. The first character in the input stream has position 0. May be called in a semantic action only.

[clearParser ()] clears the parser stack. It may be called after a parsing function has returned, to remove all pointers from the parser stack to structures that were built by semantic actions during parsing. This is not strict necessary, but reduces the memory requirements of the program.

98 PATH

Module Path

paths.

Unix:

DOS:

```
OS. Path -- SML Basis Library
exception Path
val parentArc
                  : string
                 : string
val currentArc
val fromString
                 : string -> {isAbs : bool, vol : string, arcs : string list}
                  : {isAbs : bool, vol : string, arcs : string list} -> string
val toString
val getVolume
                  : string -> string
val validVolume : {isAbs : bool, vol : string} -> bool
                  : string -> string
val getParent
                  : string -> bool
val isAbsolute
                 : string -> bool
val isRelative
                 : { path : string, relativeTo : string } -> string 
: { path : string, relativeTo : string } -> string
val mkAbsolute
val mkRelative
val concat
                  : string * string -> string
val mkCanonical : string -> string
val isCanonical : string -> bool
val splitDirFile : string -> {dir : string, file : string}
val joinDirFile : {dir : string, file : string} -> string
                  : string -> string
val dir
val file
                  : string -> string
val splitBaseExt : string -> {base : string, ext : string option}
val joinBaseExt : {base : string, ext : string option} -> string
val base : string -> string
val ext
                  : string -> string option
   This module provides OS-independent functions for manipulating
   strings that represent file names and paths in a directory
   structure. None of these functions accesses the actual filesystem.
   Definitions:
   * An arc denotes a directory or file. Under Unix or DOS, an arc may
   have form "..", ".", or "abc", or similar.
   * An absolute path has a root: Unix examples include "/", "/a/b"; DOS examples include "\", "\a\b", "A:\a\b".
   * A relative path is one without a root: Unix examples include
   "...", "a/b"; DOS examples include "...", "a\b", "A:a\b".
   ^{\star} A path has an associated volume. Under Unix, there is only one
   volume, whose name is "". Under DOS, the volume names are ""
   "A:", "C:", and similar.
   * A canonical path contains no occurrences of the empty arc "" or the current arc ".", and contains or the parent arc ".." only at
   the beginning and only if the path is relative.
   * All functions (except concat) preserve canonical paths. That is,
   if all arguments are canonical, then so will the result be.
```

* All functions are defined so that they work sensibly on canonical

* There are three groups of functions, corresponding to three ways

and

and

/d/e/f/a.b.c

A:d\e\f\a.b.c

to look at paths, exemplified by the following paths:

d/e/f/a.b.c

A:d\e\f\a.b.c

(1) A path consists of a sequence of arcs, possibly preceded by a volume and a root:

	vol	[-		arc	s]	vol	root	[-		arc	s]
Unix examples: DOS examples:	A:	-	-		a.b.c a.b	A:			_	_	a.b.c a.b

(2) A path consists of a directory part and a (last) file name part:

	directory	file	directory	file
Unix examples:	d/e/f	a.b.c	/d/e/f	a.b.c
DOS examples:	A:d\e\f	a.b	A:\d\e\f	a.b

(3) A path consists of a base and an extension:

	base	extension	base	extension
Unix examples: DOS examples:	d/e/f/a.b A:d\e\f\a	с b	/d/e/f/a.l A:\d\e\f\a	-

GROUP 0: General functions on paths:

[parentArc] is the arc denoting a parent directory: ".." under DOS and Unix.

[currentArc] is the arc denoting the current directory: "." under DOS and Unix.

[isRelative p] returns true if p is a relative path.

[isAbsolute p] returns true if p is an absolute path. Equals not (isRelative p).

[validVolume {isAbs, vol}] returns true if vol is a valid volume name for an absolute path (if isAbs=true) resp. for a relative path (if isAbs=false). Under Unix, the only valid volume name is ""; under MS DOS and MS Windows the valid volume names are "", "a:", "b:", ..., and "A:", "B:", ...

[getParent p] returns a string denoting the parent directory of p. It holds that getParent p = p if and only if p is a root.

[concat (p1, p2)] returns the path consisting of p1 followed by p2. Does not preserve canonical paths: concat("a/b", "../c") equals "a/b/../c". This is because "a/b/../c" and "a/c" may not be equivalent in the presence of symbolic links. Raises Path if p2 is not a relative path.

[mkAbsolute { path=p1, relativeTo=p2 }] returns the absolute path made by taking path p2, then p1. That is, returns p1 if p1 is absolute; otherwise returns the canonicalized concatenation of p2 and p1. Raises Path if p2 is not absolute (even if p1 is absolute).

[mkRelative { path=p1, relativeTo=p2 }] returns p1 relative to p2. That is, returns p1 if p1 is already relative; otherwise returns the relative path leading from p2 to p1. Raises Path if p2 is not absolute (and even if p1 is relative), or if p1 and p2 are both absolute but have different roots.

[mkCanonical p] returns a canonical path which is equivalent to p. Redundant occurrences of the parent arc, the current arc, and the empty arc are removed. The canonical path will never be the empty string; the empty path is converted to the current directory path ("." under Unix and DOS).

[isCanonical p] is equal to (p = mkCanonical p).

100 PATH

GROUP 1: Manipulating volumes and arcs:

[fromString p] returns {isAbs=false, vol, arcs} if the path p is relative, and {isAbs=true, vol, arcs} if the path p is absolute. In both cases vol is the volume name and arcs is the list of (possibly empty) arcs of the path. Under Unix, the volume name is always the empty string ""; under DOS it will have form "A:", "C:", or similar.

[toString path] reconstitutes a path from its root (if any) and arcs. Raises Path if applied to a relative path whose first arc is empty. It holds that toString(fromString p) = p, except that in MS DOS, slashes "/" in p will be replaced by backslashes "\". It holds that fromString (toString p) = p when no exception is raised. It holds that isRelative(toString {isAbs=false, vol, arcs}) = true when no exception is raised.

[getVolume p] returns the volume name of the path p, if given. Under Unix and MacOS, this is always the empty string "", and under MS DOS and MS Windows, it may have form "A:", "B:", ...

GROUP 2: Manipulating directory paths and file names:

[splitDirFile p] returns {dir, file} where file is the last arc in p, and dir is the path preceding that arc. A typical use is to split a path into the directory part (dir) and the filename (file).

[joinDirFile $\{dir, file\}$] returns the path p obtained by extending the path dir with the arc file.

[dir p] equals #dir (splitDirFile p).

[file p] equals #file (splitDirFile p).

GROUP 3: Manipulating file names and extensions:

[splitBaseExt s] returns {base, ext} where ext = NONE if s has no extension, and ext = SOME e if s has extension e; base is the part of s preceding the extension. A path s is considered having no extension if its last arc contains no extension separator (typically ".") or contains an extension separator only as its leftmost character, or contains an extension separator as its right-most character. Hence none of "a.b/cd", "a/.login", "a.", ".", "." and "." has an extension.

[joinBaseExt {base, ext}] returns an arc composed of the base name and the extension (if different from NONE). It is a left inverse of splitBaseExt, so joinBaseExt (splitBaseExt s) = s, but the opposite does not hold (since the extension may be empty, or may contain extension separators).

[ext s] equals #ext (splitBaseExt s).

[base s] equals #base (splitBaseExt s).

POLYGDBM 101

Module Polygdbm

Polygdbm -- GNU gdbm persistent polymorphic hashtables -- requires Dynlib

type ('key, 'data) table

exception NotFound
exception NotWriter
exception Closed
exception GdbmError of string

val withtable : string * Gdbm.openmode -> (('key, 'data) table -> 'a) -> 'a

val add : ('key, 'data) table -> 'key * 'data -> unit

val insert : ('key, 'data) table -> 'key * 'data -> unit

val find : ('key, 'data) table -> 'key -> 'data

val peek : ('key, 'data) table -> 'key -> 'data

val peek : ('key, 'data) table -> 'key -> bool

val namKey : ('key, 'data) table -> 'key -> unit

val listKeys : ('key, 'data) table -> 'key list

val numItems : ('key, 'data) table -> int

val listItems : ('key, 'data) table -> ('key, 'data) list

val app : ('key * 'data -> unit) -> ('key, 'data) table -> unit

val map : ('key * 'data -> unit) -> ('key, 'data) table -> 'a list

val fold : ('key * 'data -> 'a) -> ('key, 'data) table -> 'a

val fastwrite : bool ref

val reorganize : ('key, 'data) table -> unit

[('key, 'data) table] is the type of an opened table with keys of type 'key and associated values of type 'data. The actual values of type 'key and 'data cannot contain function closures or abstract values. Values involving references (even circular values) can be stored, but the identity of references is preserved only with every single key or value stored, not across several different values.

The Polygdbm table files of are not portable across platforms, because word size and endianness affects the lay-out of values.

A value of type table can be used only in the argument f to the withtable function. This makes sure that the table is closed after use.

[withtable (nam, mod) f] first opens the table db in file nam with mode mod, then applies f to db, then closes db. Makes sure to close db even if an exception is raised during the evaluation of f(db). Raises GdbmError with an informative message in case the table cannot be opened. E.g. the table cannot be opened for reading if already opened for writing, and cannot be opened for writing if already opened for reading.

[add db (k,v)] adds the pair (k,v) to db. Raises AlreadyThere if there is a pair $(k,_)$ in db already. Raises NotWriter if db is not opened in write mode.

[insert db (k, v)] adds the pair (k, v) to db, replacing any pair $(k, _)$ at k if present. Raises NotWriter if db is not opened in write mode.

 $[\mbox{find}(\mbox{db},\,k)\,]$ returns v if the pair $(k,\,v)$ is in db; otherwise raises NotFound.

[peek db k] returns SOME v if the pair (k, v) is in db; otherwise returns NONE.

[hasKey(db, k)] returns true if there is a pair (k, _) in db; otherwise returns false.

[remove db k] deletes the pair $(k, _)$ from the table if present; otherwise raises NotFound. Raises NotWriter if db is not opened in write mode.

102 POLYGDBM

[listKeys db] returns a list of all keys in db in an unspecified order.

[numItems db] is the number of (key, value) pairs in db. Equivalent to length(listKeys db).

[listItems db] returns a list of all (key, value) pairs in db in some order. Equivalent to

List.map (fn key => (key, find(db,key))) (listKeys db)

[app f db] is equivalent to List.app f (listItems db), provided the function f does not change the set of keys in the table. Otherwise the effect is unpredictable.

[map f db] is equivalent to List.map f (listItems db), provided the function f does not change the set of keys in the table.

Otherwise the result and effect are unpredictable.

[fold f a db] is equivalent to List.foldr (fn ((k, v), r) => f(k, v, r)) a (listItems db) provided the function f does not change the set of keys in the table. Otherwise the result and effect are unpredictable.

[fastwrite] can be set to speed up writes to a table. By default, !fastwrite is false and every write to a table will be followed by file system synchronization. This is safe, but slow if you perform thousands of writes. However, if !fastwrite is true when calling withtable, then writes may not be followed by synchronization, which may speed up writes considerably. In any case, the file system is synchronized before withtable returns.

[reorganize db] has no visible effect, but may be called after a lot of deletions to shrink the size of the table file.

POLYHASH 103

Module Polyhash

Polyhash -- polymorphic hashtables as in the SML/NJ Library

type ('key, 'data) hash_table

Polymorphic hash primitives from Caml Light

val hash : 'key -> int

val hash_param : int -> int -> 'key -> int

val mkPolyTable : int * exn -> ("_key, '_data) hash_table

[('key, 'data) hash_table] is the type of hashtables with keys of type 'key and data values of type 'data.

[mkTable (hashVal, sameKey) (sz, exc)] returns a new hashtable, using hash function hashVal and equality predicate sameKey. The sz is a size hint, and exc is the exception raised by function find. It must be the case that sameKey(k1, k2) implies hashVal(k1) = hashVal(k2) for all k1,k2.

[numItems htbl] is the number of items in the hash table.

[insert htbl (k, d)] inserts data d for key k. If k already had an item associated with it, then the old item is overwritten.

[find htbl k] returns d, where d is the data item associated with key k, or raises the exception (given at creation of htbl) if there is no such d.

[peek htbl k] returns SOME d, where d is the data item associated with key k, or NONE if there is no such d.

[peekInsert htbl (k, d)] inserts data d for key k, if k is not already in the table, returning NONE. If k is already in the table, and the associated data value is d', then returns SOME d' and leaves the table unmodified.

[remove htbl k] returns d, where d is the data item associated with key k, removing d from the table; or raises the exception if there is no such d.

[listItems htbl] returns a list of the (key, data) pairs in the hashtable.

[apply f htbl] applies function f to all (key, data) pairs in the hashtable, in some order.

[map f htbl] returns a new hashtable, whose data items have been obtained by applying f to the (key, data) pairs in htbl. The new tables have the same keys, hash function, equality predicate, and exception, as htbl.

[filter p htbl] deletes from htbl all data items which do not satisfy predicate p.

104 POLYHASH

[transform f htbl] as map, but only the (old) data values are used when computing the new data values.

[copy htbl] returns a complete copy of htbl.

[bucketSizes htbl] returns a list of the sizes of the buckets. This is to allow users to gauge the quality of their hashing function.

[hash k] returns the hash value of k, as a positive integer. If k1=k2 then hash(k1) = hash(k2), so this function can be used when creating hashtables. The application hash(k) always terminates, even on cyclic structures. (From the Caml Light implementation).

[hash_param n m k] computes a hash value for k with the same properties as for hash. The parameters n and m give more precise control over hashing. Hashing performs a depth-first, right-to-left traversal of the structure k, stopping after n meaningful nodes were encountered, or m nodes, meaningful or not, were encountered. Meaningful nodes are: integers, floating-point numbers, strings, characters, booleans, references, and constant constructors.

[mkPolyTable (sz, exc)] creates a new hashtable using the polymorphic hash function (hash) and ML equality (op =); the integer sz is a size hint and the exception exc is to be raised by find.

POSTGRES 105

Module Postgres

```
Postgres -- interface to PostgreSQL database server -- requires Dynlib
type dbconn
                                                     Connection to server
type dbresult
                                                     Result of a query
eqtype oid
                                                     Internal object id
exception Closed
                                                     Connection is closed
exception Null
                                                     Field value is NULL
Opening, closing, and maintaining database connections
                            string option, database server hoststring option, database name
val openbase : { dbhost
                   dbname
                   dboptions : string option, options
                            : string option, database server port
: string option, user passwd
: string option, tty for error log
: string option database user
                   dbport
                   dbpwd
                   dbt.t.v
                   dbuser
                 } -> dbconn
val closebase : dbconn -> unit
val db
                  : dbconn -> string
                 : dbconn -> string option
: dbconn -> string
: dbconn -> string
: dbconn -> string
val host
val options
val port
                  : dbconn -> string
val tty
                  : dbconn -> bool
val status
                  : dbconn -> unit
val reset
val errormessage : dbconn -> string option
Query execution and result set information
datatype dbresultstatus =
    Bad_response
                                An unexpected response was received
    Command ok
                                The query was a command
                                The query was "copy  from ..."
    Copy in
    Copy_out
                                The query was "copy  to ..."
    Empty_query
    Fatal_error
    Nonfatal_error
   Tuples ok
                                The query successfully returned tuples
val execute
                : dbconn -> string -> dbresult
val resultstatus : dbresult -> dbresultstatus
: dbresult -> int
: dbresult -> int
val nfields
                 : dbresult -> int -> string
: dbresult -> string vector
: dbresult -> string -> int option
val fname
val fnames
val fnumber
Accessing the fields of a resultset
                   : dbresult -> int -> int -> int
val getint
val getreal
                  : dbresult -> int -> int -> real
val getstring
val getstring
val getdate
idbresult -> int -> int -> int * int * int * Y M D
val gettime
idbresult -> int -> int -> int * int * int * H M S
val getdatetime : dbresult -> int -> int -> Date.date
val getbool
val isnull

: dbresult -> int -> int -> bool
val isnull

: dbresult -> int -> int -> bool
datatype dynval =
                                             psql bool
psql int4
    Bool of bool
    Int of int
   Real of real
                                             psql float8, float4
  String of string
                                             psql text, varchar
```

106 POSTGRES

```
Date of int * int * int
                                     psql date yyyy-mm-dd
                                     psql time hh:mm:ss
   Time of int * int * int
   DateTime of Date.date
                                      psql datetime
   Oid of oid
                                      psql oid
                                      psql bytea
   Bytea of Word8Array.array
  NullVal
                                       psql NULL
val getdynfield : dbresult -> int -> int -> dynval
val getdyntup : dbresult -> int -> dynval vector
val getdyntups : dbresult -> dynval vector vector
             : dynval -> string
val dynval2s
Bulk copying to or from a table
val copytableto : dbconn * string * (string -> unit) -> unit
val copytablefrom : dbconn * string * ((string -> unit) -> unit) -> unit
Some standard ML and Postgres types:
datatype dyntype =
                       ML bool
    BoolTy
                                               psql bool
                                               psql int4
                       ML int
   IntTy
                                               psql float8, float4
                      ML real
   RealTy
    StringTy
                     ML string
                                               psql text, varchar
                     ML (yyyy, mth, day) psql date
                     ML (hh, mm, ss)
   TimeTy
                                               psql time
                     ML Date.date
                                               psql datetime, abstime
   DateTimeTy
   OidTy
                      {\it ML} oid
                                               psql oid
   ByteArrTy
                       ML Word8Array.array psql bytea
  UnknownTy of oid
val fromtag : dyntype -> string
val ftype : dbresult -> int -> dyntype
val ftypes : dbresult -> dyntype Vector.vector
val applyto : 'a -> ('a -> 'b) -> 'b
Formatting the result of a database query as an HTML table
val formattable : dbresult -> Msp.wseq
val showquery : dbconn -> string -> Msp.wseq
```

[dbconn] is the type of connections to a PostgreSQL database.

[dbresult] is the type of result sets from SQL queries.

[oid] is the type of PostgreSQL internal object identifiers.

[openbase { dbhost, dbport, dboptions, dbtty, dbname, dbuser, dbpwd }] opens a connection to a PostgreSQL database server on the given host (default the local one) on the given port (default 5432), with the given options (default the empty string), with error logging on the given tty (default?), to the given database (defaults to the user's login name), for the given user name (defaults to the current user's login name), and the given password (default none). The result is a connection which may be used in subsequent queries.

[closebase dbconn] closes the database connection. No further queries can be executed.

[db dbconn] returns the name of the database.

[host dbconn] returns SOME h, where h is the database server host name, if the connection uses the Internet; returns NONE if the connection is to a socket on the local server.

[options dbconn] returns the options given when opening the database.

[port dbconn] returns the port number of the connection.

POSTGRES 107

[tty dbconn] returns the name of the tty used for logging.

[status dbconn] returns true if the connection is usable, false otherwise.

[reset dbconn] attempts to close and then reopen the connection to the database server.

[errormessage dbconn] returns NONE if no error occurred, and SOME msg if an error occurred, where msg describes the error.

[execute dbconn query] sends an SQL query to the database server for execution, and returns a resultset dbres.

[resultstatus dbres] returns the status of the result set dbres. After a select query that succeeded, it will be Tuples_ok.

[ntuples dbres] returns the number of tuples in the result set after a query.

[cmdtuples dbres] returns the number of tuples affected by an insert, update, or delete SQL command.

[nfields dbres] returns the number of fields in each tuple after a query.

[fname dbres fno] returns the name of field number fno (in the result set after a query). The fields are numbered $0, 1, \ldots$

[fnames dbres] returns a vector of the field names (in the result set after a query).

[fnumber dbres fname] returns SOME i where i is the number $(0, 1, \ldots)$ of the field called fname (in the result set after a query), if the result set contains such a field name; returns NONE otherwise.

[ftype dbres fno] returns the dyntype of field number fno (in the result set after a query).

[ftypes dbres] returns a vector of the dyntypes (in the result set after a query).

[fromtag dt] returns the name of the preferred PostgreSQL type used to represent values of the dyntype dt. This may be used when building 'create table' statements.

[getint dbres fno tupno] returns the integer value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[getreal dbres fno tupno] returns the floating-point value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[getstring dbres fno tupno] returns the string value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[getdate dbres fno tupno] returns the date (yyyy, mth, day) value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL. Raises Fail if the field cannot be scanned as a date.

[gettime dbres fno tupno] returns the time-of-day (hh, mm, ss) value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL. Raises Fail if the field cannot be scanned as a time.

[getdatetime dbres fno tupno] returns the Date.date value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL. Raises Fail if the field cannot be scanned as a

108 POSTGRES

date.

[getbool dbres fno tupno] returns the boolean value of field number fno in tuple tupno of result set dbres. Raises Null if the value is NULL.

[isnull dbres fno tupno] returns true if the value of field number fno in tuple tupno of result set dbres is NULL; false otherwise.

[getdynfield dbres fno tupno] returns the value of field number fno in tuple tupno of result set dbres as a dynval (a wrapped value). A NULL value is returned as NullVal. Note that the partial application (getdynfield dbres fno) precomputes the type of the field fno. Hence it is far more efficient to compute

let val getfno = getdynfield dbres fno
in tabulate(ntuples dbres, getfno) end
than to compute

let fun getfno tupno = getdynfield dbres fno tupno
in tabulate(ntuples dbres, getfno) end

because the latter repeatedly computes the type of the field.

[getdyntup dbres tupno] returns the fields of tuple tupno in result set dbres as a vector of dynvals.

[getdyntups dbres] returns all tuples of result set dbres as a vector of vectors of dynvals.

[dynval2s dv] returns a string representing the dynval dv.

[applyto x f] computes f(x). This is convenient for applying several functions (given in a list or vector) to the same value: map (applyto 5) (tabulate(3, getdynfield dbres)) equals

[getdynfield dbres 0 5, getdynfield dbres 1 5, getdynfield dbres 2 5]

[copytableto(dbconn, tablename, put)] executes a "COPY TABLE TO"
statement, applies the function put to every tuple of the table,
represented as a line of text (not terminated by newline \n), and
cleans up at the end. For instance, to copy the contents of a
table t to a text stream s (one tuple on each line), define
 fun put line =

(TextIO.output(s, line); TextIO.output(s, "\n"))
and execute
 copytableto(dbconn, "t", put).

[copytablefrom(dbconn, tablename, useput)] executes a "COPY TABLE FROM" statement, creates a put function for copying lines to the table, passes the put function to useput, and cleans up at the end. The put function may be called multiple times for each line (tuple); the end of each line is indicated with the newline character "\n" as usual. For instance, to copy the contents of a text stream s to a table t, define

copytablefrom(dbconn, "t", useput).
Note that TextIO.inputLine preserves the newline at the end of each
line.

[formattable dbresult] returns a wseq representing an HTML table. The HTML table has a column for every field in the dbresult. The first row is a table header giving the names of the fields in the dbresult. The remaining rows correspond to the tuples in the dbresult, in the order they are provided by the database server. Null fields are shown as NULL.

[showquery dbconn query] sends the SQL query to the database server, then uses formattable to format the result of the query.

PROCESS 109

Module Process

OS. Process -- SML Basis Library

type status

val success : status
val failure : status

val isSuccess : status -> bool

val system : string -> status

val atExit : (unit -> unit) -> unit

val getEnv : string -> string option

Portable functions for manipulating processes.

[success] is the unique status value that signifies successful termination of a process. Note: MS DOS (sometimes) believes that all processes are successful.

[failure] is a status value that signifies an error during execution of a process. Note that in contrast to the success value, there may be several distinct failure values. Use function isSuccess to reliably test for success.

[isSuccess sv] returns true if the status value sv represents a successful execution, false otherwise. It holds that isSuccess success = true and isSuccess failure = false.

[system cmd] asks the operating system to execute command cmd, and returns a status value.

[atExit act] registers the action act to be executed when the current SML program calls Process.exit. Actions will be executed in reverse order of registration.

[exit i] executes all registered actions, then terminates the SML process with completion code i.

[terminate i] terminates the SML process with completion code i (but without executing the registered actions).

[sleep t] suspends this process for approximately the time indicated by t. The actual time slept depends on the capabilities of the underlying system and the system load. Does not sleep at all if t <= Time.zeroTime.

[getEnv evar] returns SOME s if the environment variable evar is defined and is associated with the string s; otherwise NONE.

110 **RANDOM**

Module Random

```
Random -- random number generator
```

type generator

```
val newgenseed : real -> generator
```

val rangelist : int * int -> int * generator -> int list

[generator] is the type of random number generators, here the linear congruential generators from Paulson 1991, 1996.

[newgenseed seed] returns a random number generator with the given seed.

[newgen ()] returns a random number generator, taking the seed from the system clock.

[random gen] returns a random number in the interval [0..1).

[randomlist (n, gen)] returns a list of n random numbers in the interval [0,1).

[range (min, max) gen] returns an integral random number in the range [min, max). Raises Fail if min > max.

[rangelist (min, max) (n, gen)] returns a list of n integral random numbers in the range [min, max). Raises Fail if min > max.

RBSET 111

Module Rbset

```
Rbset -- ordered sets implemented by red-black trees
Intention: should resemble SML/NJs ORD_SET signature
signature Rbset = sig
type 'item set
exception NotFound
exception NonMonotonic
                 : ('item * 'item -> order) -> 'item set
val empty
                : ('item * 'item -> order) -> 'item -> 'item set
: 'item set * 'item -> 'item set
val singleton
val add
val add'
                  : 'item * 'item set -> 'item set
                 : 'item set * 'item list -> 'item set
val addList
                 : 'item set -> bool
: 'item set * 'item set -> bool
val isEmpty
val isSubset
                 : 'item set * 'item -> bool
val member
                  : 'item set * 'item -> 'item set
val delete
val numItems
                  : 'item set -> int
                 : 'item set -> ('item * 'item -> order)
val getOrder
                 : 'item set * 'item set -> 'item set
val union
val intersection : 'item set * 'item set -> 'item set
val difference : 'item set * 'item set -> 'item set val listItems : 'item set -> 'item list
                  : ('item -> unit) -> 'item set -> unit
val app
                  : ('item -> unit) -> 'item set -> unit
: ('item * 'b -> 'b) -> 'b -> 'item set -> 'b
val revapp
val foldr
                  : ('item * 'b -> 'b) -> 'b -> 'item set -> 'b
val foldl
                  : ('item -> 'newitem) * ('newitem * 'newitem -> order)
val map
                    -> 'item set -> 'newitem set
                  : ('item -> 'newitem) * ('newitem * 'newitem -> order)
val mapMono
                    -> 'item set -> 'newitem set
val find
                  : ('item -> bool) -> 'item set -> 'item option
                  : 'item set -> 'item option
val min
                  : 'item set -> 'item option
val max
                  : ('item -> word) -> 'item set -> word
: 'item set * 'item set -> bool
val hash
val equal
                  : 'item set * 'item set -> order
val compare
val depth
                  : 'item set -> int
datatype 'item intv =
    All
   From of 'item
    To of 'item
  | FromTo of 'item * 'item
val subset : 'item set * 'item intv -> 'item set
val sublist : 'item set * 'item intv -> 'item list
end
```

```
['item set] is the type of sets of ordered elements of type 'item. The ordering relation on the elements is used in the representation of the set. The result of combining or comparing two sets with different underlying ordering relations is undefined. The implementation uses Okasaki-style red-black trees.

[empty ordr] creates a new empty set with the given ordering relation.

[singleton ordr i] creates the singleton set containing i, with the given ordering relation.

[add(s, i)] adds item i to set s.

[addList(s, xs)] adds all items from the list xs to the set s.
```

112 RBSET

[isEmpty s] returns true if and only if the set is empty.

[equal(s1, s2)] returns true if and only if the two sets have the same elements, as determined by the ordering relation given when the sets were created.

[isSubset(s1, s2)] returns true if and only if s1 is a subset of s2.

[member(s, i)] returns true if and only if i is in s.

[delete(s, i)] removes item i from s. Raises NotFound if i is not in s.

[numItems s] returns the number of items in set s.

[union(s1, s2)] returns the union of s1 and s2.

[intersection(s1, s2)] returns the intersection of s1 and s2.

[difference(s1, s2)] returns the difference between s1 and s2 (that is, the set of elements in s1 but not in s2).

[listItems s] returns a list of the items in set s, in increasing order.

[app f s] applies function f to the elements of s, in increasing order.

[revapp f s] applies function f to the elements of s, in decreasing order.

[foldl f e s] applies the folding function f to the entries of the set in increasing order.

[foldr f e s] applies the folding function f to the entries of the set in decreasing order.

[map (f, ordr) s] creates a new set with underlying ordering ordr by applying function f to all elements of the set s.

[mapMono (f, ordr) s] creates a new set by applying the strictly monotonically increasing function f to all elements of s. The new set will have ordering ordr. This is faster than map (f, ordr) s by a logarithmic factor, but the function must satisfy $\operatorname{ordr}(f \ x, \ f \ y) = \operatorname{ordr}'(x, \ y)$

for all elements x, y in s, where ordr' is the ordering relation on s; otherwise exception NonMonotonic is thrown.

[find p s] returns SOME i, where i is an item in s which satisfies p, if one exists; otherwise returns NONE. Traverses the entries of the set in increasing order.

 $[\min \ s]$ returns SOME i, where i is the least item in the set s, if s is non-empty; returns NONE if s is empty.

[max s] returns SOME i, where i is the greatest item in the set s, if s is non-empty; returns NONE if s is empty.

[hashCode h s] returns the hashcode of the set, which is the sum of the hashcodes of its elements, as computed by the function h.

[compare (s1, s2)] returns LESS, EQUAL or GREATER according as s1 precedes, equals or follows s2 in the lexicographic ordering that would be obtained by comparing the sorted lists of elements of the two sets. It holds that

equal(s1, s2) if and only if compare(s1, s2) = EQUAL isSubset(s1, s2) implies compare(s1, s2) = LESS isSubset(s2, s1) implies compare(s1, s2) = GREATER

[subset(s, intv)] returns a set of those elements of s that belong to the interval intv. The intervals have the following meaning:

RBSET 113

```
All denotes all elements

From el denotes elements e for which cmp(e1, e) <> GREATER

To e2 denotes elements e for which cmp(e, e2) = LESS

FromTo(e1, e2) denotes elements e for which cmp(e1, e) <> GREATER

and cmp(e, e2) = LESS
```

[sublist(s, intv)] returns a list, in order, of those elements of s that belong to the interval intv. Thus sublist(s, All) is equivalent to listItems s.

114 REAL

Module Real

```
Real -- SML Basis Library
type real = real
exception Div
and Overflow
val ~
                : real -> real
               : real * real -> real
val +
                : real * real -> real
val -
val *
                : real * real -> real
val /
                : real * real -> real
val abs
                : real -> real
                : real * real -> real
val min
                : real * real -> real
val max
                : real -> int
val sign
                : real * real -> order
val compare
                : real * real -> bool
val sameSign
val toDefault
              : real -> real
val fromDefault : real -> real
                : int -> real
val fromInt
                : real -> int
val floor
val ceil
                : real -> int
val trunc
                : real -> int
val round
                : real -> int
                : real * real -> bool
val >
val >=
                : real * real -> bool
                : real * real -> bool
val <
                : real * real -> bool
val <=
                : real * real -> bool
val ==
                : real * real -> bool
val !=
val ?=
                : real * real -> bool
val toString
                : real -> string
val fromString : string -> real option
val scan : (char, 'a) StringCvt.reader -> (real, 'a) StringCvt.reader
val fmt
                 : StringCvt.realfmt -> real -> string
   [~]
   [*]
   [/]
   [+]
   [-]
   [>]
   [>=]
   [<]
   [<=] are the usual operations on defined reals (excluding NaN and Inf).
   [abs x] is x if x \ge 0, and -x if x < 0, that is, the absolute value of x.
   [\min(x, y)] is the smaller of x and y.
   [\max(x, y)] is the larger of x and y.
   [sign x] is \sim 1, 0, or 1, according as x is negative, zero, or positive.
   [compare(x, y)] returns LESS, EQUAL, or GREATER, according
   as x is less than, equal to, or greater than y.
   [sameSign(x, y)] is true iff sign x = sign y.
   [toDefault x] is x.
   [fromDefault x] is x.
   [fromInt i] is the floating-point number representing integer i.
```

REAL 115

[floor r] is the largest integer <= r (rounds towards minus infinity).
May raise Overflow.</pre>

[ceil r] is the smallest integer \geq r (rounds towards plus infinity). May raise Overflow.

[trunc r] is the numerically largest integer between r and zero (rounds towards zero). May raise Overflow.

[round r] is the integer nearest to r, using the default rounding mode. May raise Overflow.

[==(x, y)] is equivalent to x=y in Moscow ML (because of the absence of NaNs and Infs).

[!=(x, y)] is equivalent to x<>y in Moscow ML (because of the absence of NaNs and Infs).

 $\cite{figure 1.5}$ [?=(x, y)] is false in Moscow ML (because of the absence of NaNs and Infs).

[fmt spec r] returns a string representing r, in the format specified by spec (see below). The requested number of digits must be >= 0 in the SCI and FIX formats and > 0 in the GEN format; otherwise Size is raised, even in a partial application fmt(spec).

spec	description		C printf
FIX NONE FIX (SOME n) GEN NONE	fixed-point, auto choice,	12 significant digits	%e %.ne %f %.nf %.12g
GEN (SOME n)	auto choice,	n significant digits	%.nq

[toString r] returns a string representing r, with automatic choice of format according to the magnitude of r. Equivalent to (fmt (GEN NONE) r).

[fromString s] returns SOME(r) if a floating-point numeral can be scanned from a prefix of string s, ignoring any initial whitespace; returns NONE otherwise. The valid forms of floating-point numerals are described by:

```
[+\sim-]?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))([eE][+\sim-]?[0-9]+)?
```

[scan getc charsrc] attempts to scan a floating-point number from the character source charsrc, using the accessor getc, and ignoring any initial whitespace. If successful, it returns SOME(r, rest) where r is the number scanned, and rest is the unused part of the character source. The valid forms of floating-point numerals are described by:

```
[+\sim-]?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))([eE][+\sim-]?[0-9]+)?
```

116 REDBLACKMAP

Module Redblackmap

where (k, v) is an entry in m.

```
Redblackmap -- applicative maps as Red-black trees
signature Redblackmap =
sig
type ('key, 'a) dict
exception NotFound
                  : ('key * 'key -> order) -> ('key, 'a) dict
val mkDict
                 : ('key, 'a) dict * 'key * 'a -> ('key, 'a) dict
val insert
val insert : ('key, 'a) dict * 'key * 'a -> ('key, 'a) dict val find : ('key, 'a) dict * 'key -> 'a val peek : ('key, 'a) dict * 'key -> 'a option val remove : ('key, 'a) dict * 'key -> ('key, 'a) dict * 'a val numItems : ('key, 'a) dict -> int val listItems : ('key, 'a) dict -> ('key * 'a) list val app : ('key * 'a -> unit) -> ('key, 'a) dict -> unit val reverse : ('key * 'a -> unit) -> ('key, 'a) dict -> unit val reverse : ('key * 'a -> unit) -> ('key, 'a) dict -> unit val reverse : ('key * 'a -> unit) -> ('key, 'a) dict -> unit
                  : ('key * 'a -> unit) -> ('key,'a) dict -> unit
val revapp
                  : ('key * 'a * 'b -> 'b) -> 'b -> ('key, 'a) dict -> 'b
: ('key * 'a * 'b -> 'b) -> 'b -> ('key, 'a) dict -> 'b
val foldr
val foldl
                 : ('key * 'a -> 'b) -> ('key, 'a) dict -> ('key, 'b) dict
val map
val transform : ('a -> 'b) -> ('key, 'a) dict -> ('key, 'b) dict
end
    [('key, 'a) dict] is the type of applicative maps from domain type
    'key to range type 'a, or equivalently, applicative dictionaries with keys of type 'key and values of type 'a. They are implemented
    as Okasaki-style red-black trees.
    [mkDict ordr] returns a new, empty map whose keys have ordering
    ordr.
    [insert(m, i, v)] extends (or modifies) map m to map i to v.
    [find (m, k)] returns v if m maps k to v; otherwise raises NotFound.
    [peek(m, k)] returns SOME v if m maps k to v; otherwise returns NONE.
    [remove(m, k)] removes k from the domain of m and returns the
    modified map and the element v corresponding to k. Raises NotFound
    if k is not in the domain of m.
    [numItems m] returns the number of entries in m (that is, the size
    of the domain of m).
    [listItems m] returns a list of the entries (k, v) of keys k and
    the corresponding values v in m, in order of increasing key values.
    [app f m] applies function f to the entries (k, v) in m, in
    increasing order of k (according to the ordering ordr used to
    create the map or dictionary).
    [revapp f m] applies function f to the entries (k, v) in m, in
    decreasing order of k.
    [foldl f e m] applies the folding function f to the entries (k, v)
    in m, in increasing order of k.
    [foldr f e m] applies the folding function f to the entries (k, v)
    in m, in decreasing order of k.
    [map f m] returns a new map whose entries have form (k, f(k,v)),
    where (k, v) is an entry in m.
    [transform f m] returns a new map whose entries have form (k, f v),
```

REDBLACKSET 117

Module Redblackset

signature Redblackset =

sig

Redblackset -- sets implemented by Red-Black trees

```
type 'item set
exception NotFound
                 : ('item * 'item -> order) -> 'item set
                : ('item * 'item -> order) -> 'item -> 'item set
val singleton
                 : 'item set * 'item -> 'item set
: 'item set * 'item list -> 'item set
val add
val addList
                 : 'item set * 'item -> 'item
val retrieve
                 : 'item set * 'item -> 'item option
val peek
                 : 'item set -> bool
: 'item set * 'item set -> bool
val isEmpty
val equal
                 : 'item set * 'item set -> bool
val isSubset
                 : 'item set * 'item -> bool
val member
                 : 'item set * 'item -> 'item set
val delete
                 : 'item set ->
val numItems
                                  int
                 : 'item set * 'item set -> 'item set
val union
val intersection : 'item set * 'item set -> 'item set
val difference : 'item set * 'item set -> 'item set
                 : 'item set -> 'item list
val listItems
                 : ('item -> unit) -> 'item set -> unit
: ('item -> unit) -> 'item set -> unit
val app
val revapp
                 : ('item * 'b -> 'b) -> 'b -> 'item set -> 'b
val foldr
                 : ('item * 'b -> 'b) -> 'b -> 'item set -> 'b
val foldl
val find
                 : ('item -> bool) -> 'item set -> 'item option
end
   ['item set] is the type of sets of ordered elements of type 'item.
   The ordering relation on the elements is used in the representation
   of the set. The result of combining two sets with different
   underlying ordering relations is undefined. The implementation
   uses Okasaki-style Red-Black trees.
   [empty ordr] creates a new empty set with the given ordering
   relation.
   [singleton ordr i] creates the singleton set containing i, with the
   given ordering relation.
   [add(s, i)] adds item i to set s.
   [addList(s, xs)] adds all items from the list xs to the set s.
   [retrieve(s, i)] returns i if it is in s; raises NotFound otherwise.
   [peek(s, i)] returns SOME i if i is in s; returns NONE otherwise.
   [isEmpty s] returns true if and only if the set is empty.
   [equal(s1, s2)] returns true if and only if the two sets have the
   same elements.
   [isSubset(s1, s2)] returns true if and only if s1 is a subset of s2.
   [member(s, i)] returns true if and only if i is in s.
   [delete(s, i)] removes item i from s. Raises NotFound if i is not in s.
   [numItems s] returns the number of items in set s.
   [union(s1, s2)] returns the union of s1 and s2.
   [intersection(s1, s2)] returns the intersection of s1 and s2.
   [difference(s1, s2)] returns the difference between s1 and s2 (that
```

118 REDBLACKSET

is, the set of elements in s1 but not in s2).

[listItems s] returns a list of the items in set s, in increasing order.

[app f s] applies function f to the elements of s, in increasing order.

[revapp f s] applies function f to the elements of s, in decreasing order.

[foldl f e s] applies the folding function f to the entries of the set in increasing order.

[foldr f e s] applies the folding function f to the entries of the set in decreasing order.

[find p s] returns SOME i, where i is an item in s which satisfies p, if one exists; otherwise returns NONE. Traverses the entries of the set in increasing order.

REGEX 119

Module Regex

```
Regex -- regular expressions a la POSIX 1003.2 -- requires Dynlib
exception Regex of string
type regex
                              A compiled regular expression
datatype cflag =
   Extended
                              Compile POSIX extended REs
   Icase
                              Compile case-insensitive match
  Newline
                              Treat \n in target string as new line
datatype eflag =
                              Do not match ^ at beginning of string
   Notbol
                              Do not match $ at end of string
  Noteol
               : string -> cflag list -> regex
val regcomp
               : regex -> eflag list -> string -> substring vector option
val regexecBool : regex -> eflag list -> string -> bool
                : regex -> eflag list -> substring
val requexec
                  -> substring vector option
val regnexecBool : regex -> eflag list -> substring -> bool
                : { pat : string, tgt : string } -> cflag list
val regmatch
datatype replacer =
   Str of string
                                      A literal string
                                      The i'th parenthesized group
   Sus of int
   Tr of (string -> string) * int
                                      Transformation of i'th group
  Trs of substring vector -> string Transformation of all groups
                : regex -> replacer list -> string -> string
val replace1
               : regex -> replacer list -> string -> string
val replace
val substitute1 : regex -> (string -> string) -> string -> string
val substitute : regex -> (string -> string) -> string -> string
val tokens
                : regex -> string -> substring list
val fields
                : regex -> string -> substring list
val map
                : regex -> (substring vector -> 'a) -> string -> 'a list
                : regex -> (substring vector -> unit) -> string -> unit
val app
val fold
                : regex
                  -> (substring * 'a -> 'a) * (substring vector * 'a -> 'a)
                  -> 'a -> string -> 'a
```

This structure provides pattern matching with POSIX 1003.2 regular expressions.

The form and meaning of Extended and Basic regular expressions are described below. Here R and S denote regular expressions; m and n denote natural numbers; L denotes a character list; and d denotes a decimal digit:

c c Match the character c Match any character R* R* Match R zero or more times R+ R\+ Match R one or more times R S R\ S Match R or S R? R\? Match R or the empty string R\{m\} R\{m\} Match R exactly m times R\{m.\} R\{m.\} Match R at least m times	Extended	Basic	Meaning
11(,) 11(, 1) 1140011 11 40 10420 011102	R* R+ R S R?	R\+ R\ S R\?	Match any character Match R zero or more times Match R one or more times Match R or S Match R or the empty string

120 REGEX

```
R\{m,n\}
             \mathbb{R} \setminus \{m,n\}
                          Match R at least m and at most n times
                          Match any character in L
[L]
             [L]
[^L]
             [^L]
                          Match any character not in L
                          Match at string's beginning
$
             $
                          Match at string's end
(R)
             \(R\)
                          Match R as a group; save the match
                          Match the same as previous group d
\d
             ١d
                          Match \ --- similarly for *.[]^$
                         Match + --- similarly for |?{}()
```

Some example character lists L:

```
[aeiou]
                Match vowel: a or e or i or o or u
                Match digit: 0 or 1 or 2 or ... or 9
[0-9]
[^0-9]
                Match non-digit
[-+*/^]
                Match - or + or * or / or ^
                Match lowercase letter or hyphen (-)
[-a-z]
[0-9a-fA-F]
                Match hexadecimal digit
[[:alnum:]]
                Match letter or digit
[[:alpha:]]
                Match letter
[[:cntrl:]]
                Match ASCII control character
[[:digit:]]
                Match decimal digit; same as [0-9]
[[:graph:]]
                Same as [:print:] but not [:space:]
[[:lower:]]
                Match lowercase letter
[[:print:]]
                Match printable character
[[:punct:]]
                Match punctuation character
                Match SML #" ", #"\r", #"\n", #"\t", #"\v", #"\f"
[[:space:]]
[[:upper:]]
                Match uppercase letter
                Match hexadecimal digit; same as [0-9a-fA-F]
[[:xdigit:]]
[[:lower:]æøå] Match lowercase Danish letters (ISO Latin 1)
```

Remember that backslash (\) must be escaped as "\\" in SML strings.

[regcomp pat cflags] returns a compiled representation of the regular expression pat. Raises Regex in case of failure.

[cflag] is the type of compilation flags with the following meanings:

[Extended] : compile as POSIX extended regular expression.

[Icase] : compile case-insensitive match.

[Newline] : make the newline character \n significant, so ^ matches just after newline (\n), and \$ matches just before \n.

Example: Match SML integer constant:
regcomp "^~?[0-9]+\$" [Extended]

Example: Match SML alphanumeric identifier:
regcomp "^[a-zA-Z0-9][a-zA-Z0-9'_]*\$" [Extended]

Example: Match SML floating-point constant: regcomp $\[-\infty]^{-9} + (\.[0-9]+)\[-\infty]^{-9} + ($

Example: Match any HTML start tag; make the tag's name into a group:
regcomp "<([[:alnum:]]+)[^>]*>" [Extended]

[regexec regex eflags s] returns SOME(vec) if some substring of s matches regex, NONE otherwise. In case of success, vec is the match vector, a vector of substrings such that vec[0] is the (longest leftmost) substring of s matching regex, and vec[1], vec[2], ... are substrings matching the parenthesized groups in pat (numbered 1, 2, ... from left to right in the order of their opening parentheses). For a group that does not take part in the match, such as (ab) in "(ab)|(cd)" when matched against the string "xcdy", the corresponding substring is the empty substring at the beginning of the underlying string. For a group that takes part in the match repeatedly, such as the group (b+) in "(a(b+))*" when matched against "babbabbb", the corresponding substring is the last (rightmost) one matched.

[eflag] is the type of end flags with the following meaning:

[Notbol] : do not match ^ at beginning of string.

[Noteol] : do not match \$ at end of string.

[regexecBool regex eflags s] returns true if some substring of s matches regex, false otherwise. Equivalent to, but faster than, Option.isSome(regexec regexec eflags s).

[regnexec regex eflags sus] returns SOME(vec) if some substring of sus matches regex, NONE otherwise. The substrings returned in the vector vec will have the same base string as sus. Useful e.g. for splitting a string into fragments separated by substrings matching some regular expression.

[regnexecBool regex eflags sus] returns true if some substring of sus matches regex, false otherwise. Equivalent to, but faster than, Option.isSome(regnexec regexec eflags sus).

[regmatch { pat, tgt } cflags eflags] is equivalent to regexec (regcomp pat cflags) eflags tgt but more efficient when the compiled regex is used only once.

[regmatchBool { pat, tgt } cflags eflags] is equivalent to regexecBool (regcomp pat cflags) eflags tgt but more efficient when the compiled regex is used only once.

[replace regex repl s] finds the (disjoint) substrings of s
matching regex from left to right, and returns the string obtained
from s by applying the replacer list repl to every such substring
(see below). Raises Regex if it fails to make progress in
decomposing s, that is, if regex matches an empty string at the
head of s or immediately after a previous regex match.
Example use: delete all HTML tags from s:
 replace (regcomp "<[^>]+>" [Extended]) [] s

[replacel regex repl s] finds the leftmost substring b1 of s matching regex, and returns the string resulting from s by applying the replacer list repl to the match vector vec1 (see below).

Let x0 be a substring matching the entire regex and xi be the substring matching the i'th parenthesized group in regex; thus xi = vec[i] where vec is the match vector (see regexec above). Then a single replacer evaluates to a string as follows:

A replacer list repl evaluates to the concatenation of the results of the replacers. The replacers are applied from left to right.

[substitute regex f s] finds the (disjoint) substrings b1, ..., bn of s matching regex from left to right, and returns the string obtained from s by replacing every bi by f(bi). Function f is applied to the matching substrings from left to right. Raises Regex if it fails to make progress in decomposing s. Equivalent to replace regex [Tr (f, 0)] s

[substitutel regex f s] finds the leftmost substring b of s matching regex, and returns the string obtained from s by replacing that substring by f(b). Equivalent to replacel regex [Tr (f, 0)] s

[map regex f s] finds the (disjoint) substrings of s matching regex from left to right, applies f to the match vectors vec1, ..., vecn, and returns the list [f(vec1), ..., f(vecn)]. Raises Regex if it fails to make progress in decomposing s.

[app regex f s] finds the (disjoint) substrings of s matching regex from left to right, and applies f to the match vectors vec1, \dots , vecn. Raises Regex if the regex fails to make progress in decomposing s.

122 REGEX

[fields regex s] returns the list of fields in s, from left to right. A field is a (possibly empty) maximal substring of s not containing any delimiter. A delimiter is a maximal substring that matches regex. The eflags Notbol and Noteol are set. Raises Regex if it fails to make progress in decomposing s. Example use: fields (regcomp " *; *" []) "56; 23; 22;; 89; 99" [tokens regex s] returns the list of tokens in s, from left to right. A token is a non-empty maximal substring of s not containing any delimiter. A delimiter is a maximal substring that matches regex. The eflags Notbol and Noteol are set. Raises Regex if it fails to make progress in decomposing s. Equivalent to List.filter (not o Substring.isEmpty) (fields regex s) Two tokens may be separated by more than one delimiter, whereas two fields are separated by exactly one delimiter. If the only delimiter is the character #"|", then "abc||def" contains three fields: "abc" and "" and "def" "abc | def " contains two tokens: "abc" and "def" [fold regex (fa, fb) e s] finds the (disjoint) substrings b1, ..., bn of s matching regex from left to right, and splits s into the substrings a0, b1, a1, b2, a2, ..., bn, an where $n \ge 0$ and where a0 is the (possibly empty) substring of s preceding the first match, and ai is the (possibly empty) substring between the matches bi and b(i+1). Then it computes and returns fa(an, fb(vecn, ..., fa(al, fb(vecl, fa(a0, e))) ...))where veci is the match vector corresponding to bi. Raises Regex if it fails to make progress in decomposing s. If we define the auxiliary functions fun fapp f(x, r) = f x :: rfun get i vec = Substring.string(Vector.sub(vec, i)) map regex f s = List.rev (fold regex (#2, fapp f) [] s) app regex f s = fold regex (ignore, f o #1) () s fields regex s = List.rev (fold regex (op ::, #2) [] s) substitute regex f s = Substring.concat(List.rev

(fold regex (op ::, fapp (Substring.all o f o get 0)) [] s))

SML90 123

Module SML90

```
SML90 -- part of the initial basis of the 1990 Definition
Math
val sqrt : real -> real
val sin : real -> real val cos : real -> real val arctan : real -> real val exp : real -> real val ln : real -> real
Strings
              : int -> string : string -> int
val chr
val ord
val explode : string -> string list
val implode : string list -> string
exception Abs
          and Diff
          and Exp
          and Floor
          and Neg
          and Prod
          and Sum
          and Mod
          and Quot
Input/output
type instream and outstream
val std_in : instream
val open_in : string -> instream
val input : instream * int -> string
val lookahead : instream -> string
val close_in : instream -> unit
val end_of_stream : instream -> bool
```

124 SIGNAL

Module Signal

```
Signal -- SML Basis Library
eqtype signal
val abrt : signal
val alrm : signal
val bus : signal val fpe : signal
val hup : signal
val ill : signal
val int
         : signal
val kill : signal
val pipe : signal
val quit : signal
val segv : signal
val term : signal
val usr1 : signal
val usr2 : signal
val chld : signal
val cont : signal
val stop : signal
val tstp : signal
val ttin : signal
val ttou : signal
val toWord
            : signal -> Word.word
val fromWord : Word.word -> signal
   [signal] is the type of Unix/Posix-style signals, which can be sent
   to another process.
   [toWord sig] returns the signal number as an unsigned word.
   [fromWord w] returns the signal whose number is w.
   [abrt] is SIGABRT, the abort signal from abort(3).
   [alrm] is SIGALRM, a timer signal from alarm(1).
   [bus] is SIGBUS, a bus error.
   [fpe] is SIGFPE, a floating point exception.
   [hup] is SIGHUP, a hangup.
   [ill] is SIGILL, an illegal instruction.
   [int] is SIGINT, an interrupt.
   [kill] is SIGKILL, the kill signal.
   [pipe] is SIGPIPE, a broken pipe.
   [quit] is SIGQUIT, a quit from keyboard.
   [seqv] is SIGSEGV, a segmentation violation.
   [term] is SIGTERM, the termination signal.
   [usr1] is SIGUSR1, the first user signal.
   [usr2] is SIGUSR2, the second user signal.
   [chld] is SIGCHLD, child process stopped or terminated.
   [cont] is SIGCONT, continue if stopped.
   [stop] is SIGSTOP, signal to stop process.
```

SIGNAL 125

```
[tstp] is SIGTSTP, a stop signal typed at the tty.
[ttin] is SIGTTIN, tty input for background process.
[ttou] is SIGTTOU, tty output for background process.
```

Module Socket

```
Socket -- SML Basis Library -- requires Dynlib
type ('addressfam, 'socktype) sock
type 'addressfam sock_addr
Socket types
type dgram
                                         A datagram socket
type 'a stream
                                         A stream socket
type passive
                                         A passive stream
type active
                                        An active, connected, stream
Socket protocol families type pf_file
                                         The Unix file protocol family
type pf_inet
                                         The Internet protocol family
Address constructors
val fileAddr : string -> pf_file sock_addr
val inetAddr : string -> int -> pf_inet sock_addr
Socket constructors
val fileStream : unit -> (pf_file, 'a stream) sock
val fileDgram : unit -> (pf_file, dgram) sock
val inetStream : unit -> (pf_inet, 'a stream) sock
val inetDgram : unit -> (pf_inet, dgram) sock
              : ('a, passive stream) sock
-> ('a, active stream) sock * 'a sock_addr
val accept
               : ('a, 'b) sock * 'a sock_addr -> unit
val bind
               : ('a, 'b) sock * 'a sock_addr -> unit
val connect
               : ('a, passive stream) sock * int -> unit
: ('a, 'b) sock -> unit
val listen
val close
Socket management
datatype shutdown_mode =
    NO_RECVS
                                        No further receives
   NO_SENDS
                                        No further sends
  NO_RECVS_OR_SENDS
                                        No receives nor sends
val shutdown : ('a, 'b stream) sock * shutdown_mode -> unit
type sock_desc
             : ('a, 'b) sock -> sock_desc
val sockDesc
val sameDesc : sock_desc * sock_desc -> bool
               : sock_desc * sock_desc -> order
val compare
val select
    { rds : sock_desc list, wrs : sock_desc list, exs : sock_desc list,
      timeout : Time.time option }
    -> { rds : sock_desc list, wrs : sock_desc list, exs : sock_desc list }
val getinetaddr : pf_inet sock_addr -> string
Sock I/O option types
type out_flags = { don't_route : bool, oob : bool }
type in_flags = { peek : bool, oob : bool }
type 'a buf = { buf : 'a, ofs : int, size : int option }
Socket output operations
               : ('a, active stream) sock * Word8Vector.vector buf -> int
val sendVec
               : ('a, active stream) sock * Word8Array.array buf -> int
val sendArr
               : ('a, active stream) sock * Word8Vector.vector buf
val sendVec'
                 * out_flags -> int
               : ('a, active stream) sock * Word8Array.array buf
val sendArr'
                 * out_flags -> int
val sendVecTo : ('a, dgram) sock * 'a sock_addr * Word8Vector.vector buf
                  -> int
val sendArrTo : ('a, dgram) sock * 'a sock_addr * Word8Array.array buf
```

```
-> int
val sendVecTo': ('a, dgram) sock * 'a sock_addr * Word8Vector.vector buf
                  * out_flags -> int
val sendArrTo' : ('a, dgram) sock * 'a sock_addr * Word8Array.array buf
                  * out_flags -> int
Socket input operations
                 : ('a, active stream) sock * int -> Word8Vector.vector
: ('a, active stream) sock * Word8Array.array buf -> int
val recvVec
val recvArr
                  : ('a, active stream) sock * int * in_flags
val recvVec'
                    -> Word8Vector.vector
                  : ('a, active stream) sock * Word8Array.array buf * in_flags
val recvArr'
                     -> int
val recvVecFrom : ('a, dgram) sock * int
                    -> Word8Vector.vector * 'a sock_addr
val recvArrFrom : ('a, dgram) sock * Word8Array.array buf
                    -> int * 'a sock_addr
val recvVecFrom' : ('a, dgram) sock * int * in_flags
                    -> Word8Vector.vector * 'a sock addr
val recvArrFrom' : ('a, dgram) sock * Word8Array.array buf * in_flags
                    -> int * 'a sock_addr
```

Structure Socket defines functions for creating and using sockets, a means for communication between SML processes on the same machine or via a network.

[('addressfam, 'socktype) sock] is the type of sockets with address family 'addressfam and having type 'socktype.

['addressfam sock_addr] is the type of sockets addresses.

The possible address (protocol) families are

The possible socket types are

type dgram datagram sockets type 'a stream stream sockets

type passive passive stream sockets

type active active, or connected, stream sockets

[fileAddr fname] returns a socket address for the Unix protocol family, created from the given file name fname.

[inetAddr inetaddr portno] returns a socket address for the Internet protocol family, created from the given Internet number (e.g. "130.225.40.253") and port number (e.g. 8080).

[fileStream ()] returns a new stream socket for the Unix protocol family.

[fileDgram ()] returns a new datagram socket for the Unix protocol family.

[inetStream ()] returns a new stream socket for the Internet protocol family.

[inetDgram ()] returns a new datagram socket for the Internet protocol family.

[accept sock] extracts the first connection on the queue of pending connections to sock. Returns (sock', addr) where sock' is a copy of the socket sock, bound to that connection, and addr is the address of the communications counterpart (the other end of the connection). Blocks if no connections are pending. The stream socket sock must have been assigned a name (with bind) and must be listening for connections (following a call to listen).

[bind sock addr] binds the socket sock to the address addr, that is, assigns the name addr to the socket. Binding a name in the

Unix protocol family creates a socket in the file system that must be deleted when it is no longer needed

[connect (sock, addr)] attempts to connect socket sock to the communications peer at address addr. If sock is a datagram socket, then addr is the address to which datagrams is to be sent, and the only address from which datagrams will be accepted. If sock is a stream socket, then addr specifies another socket to which to connect.

[listen (sock, queuelen)] enables the passive stream socket sock to accept incoming connections. The parameter queuelen specifies the maximal number of pending connections. Further connections from clients may be refused when this limit is reached.

[close sock] closes the socket.

[shutdown sock shutdown_mode] shuts down socket sock for further communication, as specified by the shutdown mode parameter:

[NO_RECVS] no further receives are allowed;

[NO_SENDS] no further sends are allowed;

[NO_RECVS_OR_SENDS] no further receives or sends are allowed.

[getinetaddr addr] returns the Internet number (e.g. "130.225.40.253") of the Internet socket address addr.

['a buf] is the type of records { buf, ofs, size } which represent subvectors or subarrays:

if size = SOME s it represents buf[ofs..ofs+s-1];

if size = NONE it represents buf[ofs..len-1] where len is buf's length. When the subbuffer is used in a call, exception Subscript will be raised if ofs < 0 or size < 0 or ofs+size > len.

[sendVec (sock, vecbuf)] transmits the bytes from buffer vecbuf on the active stream socket sock. Returns the number of bytes sent. Blocks until sufficient space is available at the socket.

[sendArr (sock, arrbuf)] is analogous til sendVec.

[sendVec' (sock, vecbuf, out_flags)] transmits the bytes from buffer vecbuf on the active stream socket sock, observing the out_flags. Returns the number of bytes sent. Blocks until sufficient space is available at the socket.

[out_flags] is the type of records { don't_route, oob } in which the field don't_route specifies whether routing should be bypassed, and the field oob specifies whether data should be sent out-of-band.

[sendArr' (sock, arrbuf, out_flags)] is analogous til sendVec'.

[sendVecTo (sock, addr, vecbuf)] transmits the bytes from buffer vecbuf on the datagram socket sock to the target address addr. Returns the number of bytes sent. Blocks until sufficient space is available at the socket.

[sendArrTo (sock, addr, arrbuf)] is analogous til sendVecTo.

[sendVecTo' (sock, addr, vecbuf, out_flags)] transmits the bytes from buffer vecbuf on the datagram socket sock to the target address addr, observing the out_flags. Returns the number of bytes sent. Blocks until sufficient space is available at the socket. See above for a description of vecbuf and out_flags.

[sendArrTo' (sock, addr, arrbuf, out_flags)] is analogous til sendVecTo'.

[recvVec (sock, n)] receives up to n bytes from the active stream socket sock. Returns a byte vector containing the bytes actually received. Blocks until some data become available at the socket, then returns any available data, up to n bytes. Excess data are

not lost; they are available for subsequent receive calls.

[recvArr (sock, arrbuf)] receives bytes from the active stream socket sock into the subarray arrbuf, up to the available space. If #size(arrbuf) = SOME(s) the available space is s bytes; if #size(arrbuf) = NONE the available space is len - #ofs(arrbuf) bytes. Returns the number of bytes actually received. Blocks until some data become available at the socket. Excess data are not lost; they are available for subsequent receive calls.

[recvVec' (sock, n, in_flags)] receives up to n bytes from the active stream socket sock, observing the in_flags. Returns a byte vector containing the bytes actually received. Blocks until some data become available at the socket, then returns any available data, up to n bytes. Data in excess of n bytes are not lost; they are available for subsequent receive calls.

[in_flags] is the type of records { peek, oob } in which the field peek specifies that the data read should not be removed from the receive queue, and the field oob specifies that data may be received out-of-band.

[recvArr' (sock, arrbuf, in_flags)] receives bytes from the active stream socket sock into the subarray arrbuf, observing the in_flags, up to the available space. Returns the number of bytes actually received. Blocks until some data become available at the socket. Excess data are not lost; they are available for subsequent receive calls.

[recvVecFrom (sock, n)] receives up to n bytes from the datagram socket sock. Returns a byte vector containing the bytes actually received. Blocks until some data become available at the socket, then returns any available data, up to n bytes.

[recvArrFrom (sock, arrbuf)] receives bytes from the datagram socket sock into the subarray arrbuf. Returns the number of bytes actually received. Blocks until some data become available at the socket

[recvVecFrom' (sock, n, in_flags)] receives up to n bytes from the datagram socket sock, observing the in_flags (see above). Returns (vec, addr) where vec is a byte vector containing the bytes actually received, and addr is the source address of the message. Blocks until some data become available at the socket, then returns any available data, up to n bytes.

[recvArrFrom' (sock, arrbuf, in_flags)] receives bytes from the datagram socket sock into the array buffer arrbuf, observing the in_flags (see above). Returns (n, addr) where n is the number of bytes actually received, and addr is the source address of the message. Blocks until some data become available at the socket.

[sockDesc sock] returns a descriptor for the socket sock, to be used in a call to select.

[compare (sd1, sd2)] compares sd1 and sd2 according to an unspecified total ordering, and returns LESS if sd1 precedes sd2, returns GREATER is sd1 precedes sd2, and returns EQUAL otherwise.

[sameDesc (sd1, sd2)] returns true if sd1 and sd2 describe the same socket. Equivalent to compare(sd1, sd2) = EQUAL.

[select { rds, wrs, exs, timeout }] blocks the calling process until some input/output operations become possible on some sockets. The call will check the sockets described in rds for reading, those in wrs for writing, and those in exs for exceptional conditions. Returns { rds, wrs, exs } where rds now is a list of descriptors of sockets ready for reading, wrs are ready for writing, and exs have exceptional conditions. The order of the socket descriptors in the results is the same as their order in the corresponding arguments. If timeout is NONE then the call blocks until some input/output operations become possible; if timeout is SOME(t) then the call

blocks for at most time t.

A server socket is considered ready for reading if there is a pending connection which can be accepted with 'accept'. A client socket is ready for writing when its connection is fully established.

SPLAYMAP 131

Module Splaymap

Splaymap -- applicative maps implemented by splay-trees From SML/NJ lib 0.2, copyright 1993 by AT&T Bell Laboratories type ('key, 'a) dict exception NotFound : ('_key * '_key -> order) -> ('_key, '_a) dict : ('_key, '_a) dict * '_key * '_a -> ('_key, '_a) dict val mkDict val insert : ('_key, '_a) dict * '_key * '_a -> ('_key, '_a) dict
val find : ('key, 'a) dict * 'key -> 'a
val peek : ('key, 'a) dict * 'key -> 'a option
val remove : ('_key, '_a) dict * '_key -> ('_key, '_a) dict * '_a
val numItems : ('key, 'a) dict -> int
val listItems : ('key, 'a) dict -> ('key * 'a) list
val app : ('key * 'a -> unit) -> ('key, 'a) dict -> unit
val revapp : ('key * 'a -> 'b) -> ('key, 'a) dict -> 'b
val foldl : ('key * 'a * 'b -> 'b) -> 'b -> ('key, 'a) dict -> 'b
val map : ('key * 'a -> 'b) -> 'b -> ('key, 'a) dict -> 'b
val map : ('key * 'a -> 'b) -> 'b -> ('key, 'a) dict -> 'b val insert val map : ('_key * 'a -> '_b) -> ('_key,'a) dict -> ('_key, '_b) dict val transform : ('a -> '_b) -> ('_key,'a) dict -> ('_key, '_b) dict [('key, 'a) dict] is the type of applicative maps from domain type 'key to range type 'a, or equivalently, applicative dictionaries with keys of type 'key and values of type 'a. They are implemented as ordered splay-trees (Sleator and Tarjan). [mkDict ordr] returns a new, empty map whose keys have ordering [insert(m, i, v)] extends (or modifies) map m to map i to v. [find (m, k)] returns v if m maps k to v; otherwise raises NotFound. [peek(m, k)] returns SOME v if m maps k to v; otherwise returns NONE. [remove(m, k)] removes k from the domain of m and returns the modified map and the element v corresponding to k. Raises NotFound if k is not in the domain of m. [numItems m] returns the number of entries in m (that is, the size of the domain of m). [listItems m] returns a list of the entries (k, v) of keys k and the corresponding values v in m, in increasing order of k. [app f m] applies function f to the entries (k, v) in m, in increasing order of k (according to the ordering ordr used to create the map or dictionary). [revapp f m] applies function f to the entries (k, v) in m, in decreasing order of k. [foldl f e m] applies the folding function f to the entries (k, v)in m, in increasing order of k. [foldr f e m] applies the folding function f to the entries (k, v)in m, in decreasing order of k. [map f m] returns a new map whose entries have form (k, f(k,v)), where (k, v) is an entry in m. [transform f m] returns a new map whose entries have form (k, f v),

where (k, v) is an entry in m.

132 SPLAYSET

Splayset -- applicative sets implemented by splay-trees

Module Splayset

```
From SML/NJ lib 0.2, copyright 1993 by AT&T Bell Laboratories
type 'item set
exception NotFound
                                 : ('_item * '_item -> order) -> '_item set
: ('_item * '_item -> order) -> '_item -> '_item set
: '_item set * '_item -> '_item set
: '_item set * '_item list -> '_item set
val empty
val singleton
val add
val addList
                                  : 'item set * 'item -> 'item
val retrieve
                                  : 'item set * 'item -> 'item option
val peek
                                  : 'item set -> bool
: 'item set * 'item set -> bool
val isEmpty
val equal
                                 : 'item set * 'item set -> bool
val isSubset
                                  : 'item set * 'item -> bool
val member
                                          _item set * '_item -> '_item set
val delete
val delete : __item set __item set val numItems : 'item set -> int val union : '_item set * '_item set -> '_item set val intersection : '_item set * '_item set -> '_item set val difference : '_item set * '_item set -> '_item set val difference : '_item set * '_item set -> '_item se
                                  : 'item set -> 'item list
val listItems
                                  : ('item -> unit) -> 'item set -> unit
: ('item -> unit) -> 'item set -> unit
val app
val revapp
                                  : ('item * 'b -> 'b) -> 'b -> 'item set -> 'b
val foldr
                                  : ('item * 'b -> 'b) -> 'b -> 'item set -> 'b
val foldl
val find
                                  : ('item -> bool) -> 'item set -> 'item option
      ['item set] is the type of sets of ordered elements of type 'item.
      The ordering relation on the elements is used in the representation
      of the set. The result of combining two sets with different
      underlying ordering relations is undefined. The implementation
      uses splay-trees (Sleator and Tarjan).
      [empty ordr] creates a new empty set with the given ordering
      relation.
      [singleton ordr i] creates the singleton set containing i, with the
      given ordering relation.
      [add(s, i)] adds item i to set s.
      [addList(s, xs)] adds all items from the list xs to the set s.
       [retrieve(s, i)] returns i if it is in s; raises NotFound otherwise.
       [peek(s, i)] returns SOME i if i is in s; returns NONE otherwise.
      [isEmpty s] returns true if and only if the set is empty.
       [equal(s1, s2)] returns true if and only if the two sets have the
      same elements.
      [isSubset(s1, s2)] returns true if and only if s1 is a subset of s2.
      [member(s, i)] returns true if and only if i is in s.
       [delete(s, i)] removes item i from s. Raises NotFound if i is not in s.
       [numItems s] returns the number of items in set s.
       [union(s1, s2)] returns the union of s1 and s2.
       [intersection(s1, s2)] returns the intersection of s1 and s2.
       [difference(s1, s2)] returns the difference between s1 and s2 (that
      is, the set of elements in s1 but not in s2).
```

SPLAYSET 133

[listItems s] returns a list of the items in set s, in increasing order.

[app f s] applies function f to the elements of s, in increasing order.

[revapp f s] applies function f to the elements of s, in decreasing order.

[foldl f e s] applies the folding function f to the entries of the set in increasing order.

[foldr f e s] applies the folding function f to the entries of the set in decreasing order.

[find p s] returns SOME i, where i is an item in s which satisfies p, if one exists; otherwise returns NONE.

134 STRING

Module String

```
String -- SML Basis Library
local
    type char = Char.char
in
    type string = string
    val maxSize : int
    val size
                    : string -> int
                   : string * int -> char
    val substring : string * int * int -> string
    val extract
                    : string * int * int option -> string
    val ^
                  : string * string -> string
    val concat
                   : string list -> string
    val concatWith : string -> string list -> string
val str : char -> string
                 : char list -> string
    val implode
                   : string -> char list
    val explode
                    : (char -> char) -> string -> string
    val map
    val translate : (char -> string) -> string -> string
    val tokens : (char -> bool) -> string -> string list val fields : (char -> bool) -> string -> string list
    val isPrefix : string -> string -> bool
val isSuffix : string -> string -> bool
    val isSubstring : string -> string -> bool
    val fromString : string -> string option
                                                  ML escape sequences
    val toString
                   : string -> string
                                                  ML escape sequences
    val fromCString : string -> string option
                                                  C escape sequences
    val toCString : string -> string
                                                  C escape sequences
    val < : string * string -> bool
    val <= : string * string -> bool
    val > : string * string -> bool
    val >= : string * string -> bool
   [string] is the type of immutable strings of characters, with
   constant-time indexing.
   [maxSize] is the maximal number of characters in a string.
   [size s] is the number of characters in string s.
   [sub(s, i)] is the i'th character of s, counting from zero.
  Raises Subscript if i<0 or i>=size s.
   [substring(s, i, n)] is the string s[i..i+n-1]. Raises Subscript
   if i<0 or n<0 or i+n>size s. Equivalent to extract(s, i, SOME n).
   [extract (s, i, NONE)] is the string s[i..size s-1].
  Raises Subscript if i<0 or i>size s.
   [extract (s, i, SOME n)] is the string s[i..i+n-1].
  Raises Subscript if i<0 or n<0 or i+n>size s.
   [s1 ^ s2] is the concatenation of strings s1 and s2.
   [concat ss] is the concatenation of all the strings in ss.
  Raises Size if the sum of their sizes is greater than maxSize.
   [concatWith sep ss] is the concatenation of all the strings in ss,
   using sep as a separator. Thus
      concatWith sep ss
                                    is the empty string ""
      concatWith sep [s]
                                    is s
```

concatWith sep $[s1, \ldots, sn]$ is concat $[s1, sep, \ldots, sep, sn]$. Raises Size if the resulting string would have more than maxSize characters.

[str c] is the string of size one which contains the character c.

[implode cs] is the string containing the characters in the list cs. Equivalent to concat (List.map str cs).

[explode s] is the list of characters in the string s.

[map f s] applies f to every character of s, from left to right, and returns the string consisting of the resulting characters. Equivalent to CharVector.map f s

and to implode (List.map f (explode s)).

[translate f s] applies f to every character of s, from left to right, and returns the concatenation of the resulting strings. Raises Size if the sum of their sizes is greater than maxSize. Equivalent to concat (List.map f (explode s)).

[tokens p s] returns the list of tokens in s, from left to right, where a token is a non-empty maximal substring of s not containing any delimiter, and a delimiter is a character satisfying p.

[fields p s] returns the list of fields in s, from left to right, where a field is a (possibly empty) maximal substring of s not containing any delimiter, and a delimiter is a character satisfying p.

Two tokens may be separated by more than one delimiter, whereas two fields are separated by exactly one delimiter. If the only delimiter is the character $\# \| \|$, then

"abc||def" contains two tokens: "abc" and "def"
"abc||def" contains three fields: "abc" and "" and "def"

[isPrefix s1 s2] is true if s1 is a prefix of s2. That is, if there exists a string u such that s1 $^{\circ}$ u = s2.

[isSuffix s1 s2] is true if s1 is a suffix of s2. That is, if there exists a string t such that $t ^s = s2$.

[isSubstring s1 s2] is true if s1 is a substring of s2. That is, if there exist strings t and u such that t s s1 s u = s2.

[fromString s] scans the string s as an ML source program string, converting escape sequences into the appropriate characters. Does not skip leading whitespace.

[toString s] returns a string corresponding to s, with non-printable characters replaced by ML escape sequences. Equivalent to String.translate Char.toString.

[fromCString s] scans the string s as a C source program string, converting escape sequences into the appropriate characters. Does not skip leading whitespace.

[toCString s] returns a string corresponding to s, with non-printable characters replaced by C escape sequences. Equivalent to String.translate Char.toCString.

[compare (s1, s2)] does lexicographic comparison, using the standard ordering Char.compare on the characters. Returns LESS, EQUAL, or GREATER, according as s1 is less than, equal to, or greater than s2.

[collate cmp (s1, s2)] performs lexicographic comparison, using the given ordering cmp on characters.

[<]

[<=]

[>]

136 STRING

 $\ensuremath{[>=]}$ compare strings lexicographically, using the representation ordering on characters.

STRINGCVT 137

Module StringCvt

```
StringCvt -- SML Basis Library
datatype radix = BIN | OCT | DEC | HEX
datatype realfmt =
    SCI of int option
                          scientific, arg = # dec. digits, dflt=6
    FIX of int option fixed-point, arg = # dec. digits, dflt=6
   GEN of int option auto choice of the above,
                           arg = # significant digits, dflt=12
                           character source state
type cs
type ('a, 'b) reader = 'b -> ('a * 'b) option
val scanString : ((char, cs) reader -> ('a, cs) reader) -> string -> 'a option
                : (char -> bool) -> (char, 'a) reader -> 'a -> string * 'a
: (char -> bool) -> (char, 'a) reader -> 'a -> string
: (char -> bool) -> (char, 'a) reader -> 'a -> 'a
: (char, 'a) reader -> 'a -> 'a
val splitl
val takel
val dropl
val skipWS
val padLeft
                : char -> int -> string -> string
val padRight : char -> int -> string -> string
```

This structure presents tools for scanning strings and values from functional character streams, and for simple formatting.

```
[('elm, 'src) reader] is the type of source readers for reading a sequence of 'elm values from a source of type 'src. For instance, a character source reader \[
```

getc : (char, cs) reader

is used for obtaining characters from a functional character source src of type cs, one at a time. It should hold that

A character source scanner takes a character source reader getc as argument and uses it to scan a data value from the character source.

[scanString scan s] turns the string s into a character source and applies the scanner 'scan' to that source.

[split1 p getc src] returns (pref, suff) where pref is the longest prefix (left substring) of src all of whose characters satisfy p, and suff is the remainder of src. That is, the first character retrievable from suff, if any, is the leftmost character not satisfying p. Does not skip leading whitespace.

[takel p getc src] returns the longest prefix (left substring) of
src all of whose characters satisfy predicate p. That is, if the
left-most character does not satisfy p, the result is the empty
string. Does not skip leading whitespace. It holds that
 takel p getc src = #1 (splitl p getc src)

[dropl p getc src] drops the longest prefix (left substring) of
src all of whose characters satisfy predicate p. If all characters
do, it returns the empty source. It holds that
 dropl p getc src = #2 (splitl p getc src)

[skipWS getc src] drops any leading whitespace from src. Equivalent to dropl Char.isSpace.

[padLeft c n s] returns the string s if size s >= n, otherwise pads s with (n - size s) copies of the character c on the left. In other words, right-justifies s in a field n characters wide.

138 STRINGCVT

[padRight c n s] returns the string s if size s >= n, otherwise pads s with (n - size s) copies of the character c on the right. In other words, left-justifies s in a field n characters wide.

SUBSTRING 139

Module Substring

Substring -- SML Basis Library type substring : string * int * int -> substring
: string * int * int option -> substring val substring val extract : string -> substring val full : string -> substring val all val string : substring -> string : substring -> (string * int * int) val base val isEmpty : substring -> bool : substring -> (cnar substring : substring -> char option : int -> substring -> substring : int -> substring -> substring : substring * int -> char : substring -> int : substring * int * int option : substring -> (char * substring) option val getc val first val triml val trimr val sub val size : substring * int * int option -> substring
: substring list -> string val slice val concat val concatWith : string -> substring list -> string : substring -> char list val collate : (char * char -> order) -> substring * substring -> order val dropl : (char -> bool) -> substring -> substring val dropr : (char -> bool) -> substring -> substring val takel : (char -> bool) -> substring -> substring : (char -> bool) -> substring -> substring val taker : (char -> bool) -> substring -> substring * substring
: (char -> bool) -> substring -> substring * substring
: substring * int -> substring * substring val splitl val splitr val splitAt : string -> substring -> substring * substring
: string -> substring -> bool
: string -> substring -> bool val position val isPrefix val isSuffix val isSubstring : string -> substring -> bool exception Span val span : substring * substring -> substring val translate : (char -> string) -> substring -> string : (char -> bool) -> substring -> substring list val tokens val fields : (char -> bool) -> substring -> substring list val foldl : (char * 'a -> 'a) -> 'a -> substring -> 'a : (char * 'a -> 'a) -> 'a -> substring -> 'a : (char -> unit) -> substring -> unit val foldr val app [substring] is the type of substrings of a basestring, an efficient representation of a piece of a string. A substring (s,i,n) is valid if $0 \le i \le i+n \le size s$, or equivalently, 0 <= i and 0 <= n and i+n <= size s. A valid substring (s, i, n) represents the string s[i...i+n-1]. Invariant in the implementation: Any value of type substring is valid. A substring is the same as a CharVectorSlice.slice, so substrings may be processed using the functions declared in CharVectorSlice. [substring(s, i, n)] creates the substring (s, i, n), consisting of the substring of s with length n starting at i. Raises Subscript if i<0 or n<0 or i+n > size s. Equivalent to extract(s, i, SOME n). [extract(s, i, NONE)] creates the substring (s, i, size s-i) consisting of the tail of s starting at i. Raises Subscript if i<0 or i > size s.

[extract(s, i, SOME n)] creates the substring (s, i, n),

140 SUBSTRING

consisting of the substring of s with length n starting at i. Raises Subscript if i<0 or n<0 or i+n > size s. [full s] is the substring (s, 0, size s). [all s] is the same as full(s). Its use is deprecated. [string sus] is the string s[i..i+n-1] represented by sus = (s, i, n). [base sus] is the concrete triple (s, i, n), where sus = (s, i, n). [isEmpty (s, i, n)] true if the substring is empty (that is, n = 0). [getc sus] returns SOME(c, rst) where c is the first character and rst the remainder of sus, if sus is non-empty; otherwise returns NONE. Note that #1 o valOf o scanFn Substring.getc is equivalent to, but more efficient than, valOf o StringCvt.scanString scanFn o Substring.string [first sus] returns SOME c where c is the first character in sus, if sus is non-empty; otherwise returns NONE. [triml k sus] returns sus less its leftmost k characters; or the empty string at the end of sus if it has less than k characters. Raises Subscript if k < 0, even in the partial application triml(k). [trimr k sus] returns sus less its rightmost k characters; or the empty string at the beginning of sus if it has less than k characters. Raises Subscript if k < 0, even in the partial application triml(k). [sub (sus, k)] returns the k'th character of the substring; that is, s(i+k) where sus = (s, i, n). Raises Subscript if k<0 or k>=n. [size (s, i, n)] returns the size of the substring, that is, n. [slice (sus, i', NONE)] returns the substring (s, i+i', n-i'), where sus = (s, i, n). Raises Subscript if i' < 0 or i' > n. [slice (sus, i', SOME n')] returns the substring (s, i+i', n'), where sus = (s, i, n). Raises Subscript if i' < 0 or n' < 0 or i'+n' > n. [concat suss] returns a string consisting of the concatenation of the substrings. Equivalent to String.concat (List.map string suss). Raises Size if the resulting string would be longer than String.maxSize. [concatWith sep suss] returns a string consisting of the concatenation of the substrings in suss, using sep as a separator. Equivalent to String.concatWith sep (List.map string suss). Raises Size if the resulting string would be longer than String.maxSize. [explode sus] returns the list of characters of sus, that is, [s(i), s(i+1), ..., s(i+n-1)]where sus = (s, i, n). Equivalent to String.explode(string ss). [compare (sus1, sus2)] performs lexicographic comparison, using the standard ordering Char.compare on the characters. Returns LESS, EQUAL, or GREATER, according as susl is less than, equal to, or greater than sus2. Equivalent to, but more efficient than, String.compare(string sus1, string sus2). [collate cmp (sus1, sus2)] performs lexicographic comparison, using the given ordering cmp on characters. Equivalent to, but more efficient than, String.collate cmp (string sus1, string sus2). [dropl p sus] drops the longest prefix (left substring) of sus all of whose characters satisfy predicate p. If all characters do, it returns the empty substring (s, i+n, 0) where sus = (s, i, n).

[dropr p sus] drops the longest suffix (right substring) of sus all of whose characters satisfy predicate p. If all characters do, it returns the empty substring (s, i, 0) where sus = (s, i, n).

[takel p sus] returns the longest prefix (left substring) of sus all of whose characters satisfy predicate p. That is, if the left-most character does not satisfy p, returns the empty (s, i, 0) where sus = (s, i, n).

[taker p sus] returns the longest suffix (right substring) of sus all of whose characters satisfy predicate p. That is, if the right-most character satisfies p, returns the empty (s, i+n, 0) where sus = (s, i, n).

Let p be a predicate and xxxxfyyyyfzzzz a string where all characters in xxxx and zzzz satisfy p, and f a is character not satisfying p. Then

```
sus = xxxxfyyyyfzzzz sus = xxxxzzzz
dropl p sus = fyyyyfzzzz
```

dropr p sus = xxxxfyyyyf

takel p sus = xxxx xxxzzzzz taker p sus = zzzz xxxxzzzzz

It also holds that

concat[takel p sus, dropl p sus] = string sus concat[dropr p sus, taker p sus] = string sus

[split1 p sus] splits sus into a pair (sus1, sus2) of substrings where sus1 is the longest prefix (left substring) all of whose characters satisfy p, and sus2 is the rest. That is, sus2 begins with the leftmost character not satisfying p. Disregarding sideeffects, we have:

splitl p sus = (takel p sus, dropl p sus).

[splitr p sus] splits sus into a pair (sus1, sus2) of substrings where sus2 is the longest suffix (right substring) all of whose characters satisfy p, and sus1 is the rest. That is, sus1 ends with the rightmost character not satisfying p. Disregarding sideeffects, we have:

splitr p sus = (dropr p sus, taker p sus)

[splitAt (sus, k)] returns the pair (sus1, sus2) of substrings, where sus1 contains the first k characters of sus, and sus2 contains the rest. Raises Subscript if k < 0 or k > size sus.

[isPrefix s1 s2] is true if s1 is a prefix of s2. That is, if there exists a string u such that $s1 ^ u = string s2$.

[isSuffix s1 s2] is true if s1 is a suffix of s2. That is, if there exists a string t such that $t ^s 1 = string s2$.

[isSubstring s1 s2] is true if s1 is a substring of s2. That is, if there exist strings t and u such that $t ^s1 ^u = string s2$.

[position s (s',i,n)] splits the substring into a pair (pref, suff) of substrings, where suff is the longest suffix of (s',i,n) which has s as a prefix. More precisely, let $m=size\ s$. If there is a least index k in i..i+n-m for which s=s'[k..k+m-1], then the result is pref=(s',i,k-i) and suff = (s',k,n-(k-i)); otherwise the result is pref=(s',i,n) and suff = (s',i+n,0).

[span (sus1, sus2)] returns a substring spanning from the start of sus1 to the end of sus2, provided this is well-defined: sus1 and sus2 must have the same underlying string, and the start of sus1 must not be to the right of the end of sus2; otherwise raises Span.

More precisely, if base(sus1) = (s,i,n) and base(sus2) = (s',i',n') and s = s' and i <= i'+n', then base(join(sus1, sus2)) = (s,i,i'+n'-i). This may be used to compute 'span', 'union', and 'intersection'.

[translate f sus] applies f to every character of sus, from left to right, and returns the concatenation of the results. Raises Size if the sum of their sizes is greater than String.maxSize.

142 SUBSTRING

Equivalent to String.concat(List.map f (explode sus)).

[tokens p sus] returns the list of tokens in sus, from left to right, where a token is a non-empty maximal substring of sus not containing any delimiter, and a delimiter is a character satisfying p.

[fields p sus] returns the list of fields in sus, from left to right, where a field is a (possibly empty) maximal substring of sus not containing any delimiter, and a delimiter is a character satisfying p.

Two tokens may be separated by more than one delimiter, whereas two fields are separated by exactly one delimiter. If the only delimiter is the character $\#\|\|$, then

```
"abc | def" contains two tokens: "abc" and "def"
"abc | def" contains three fields: "abc" and "" and "def"
```

[foldl f e sus] folds f over sus from left to right. That is, evaluates $f(s[i+n-1], f(\ldots, f(s[i+1], f(s[i] % e)) \ldots))$ tail-recursively, where sus = (s, i, n). Equivalent to List.foldl f e (explode sus).

[foldr f e sus] folds f over sus from right to left. That is, evaluates f(s[i], f(s[i+1], f(..., f(s[i+n-1] % e) ...))) tail-recursively, where sus = (s, i, n). Equivalent to List.foldr f e (explode sus).

[app f sus] applies f to all characters of sus, from left to right. Equivalent to List.app f (explode sus).

SUSP 143

Module Susp

Susp -- support for lazy evaluation
type 'a susp
val delay : (unit -> 'a) -> 'a susp
val force : 'a susp -> 'a

 $\mbox{['a susp]}$ is the type of lazily evaluated expressions with result type 'a.

[delay (fn () => e)] creates a suspension for the expression e. The first time the suspension is forced, the expression e will be evaluated, and the result stored in the suspension. All subsequent forcing of the suspension will just return this result, so e is evaluated at most once. If the suspension is never forced, then e is never evaluated.

[force su] forces the suspension su and returns the result of the expression e stored in the suspension.

144 TEXTIO

Module TextIO

```
TextIO -- SML Basis Library
type elem = Char.char
type vector = string
Text input
type instream
val openIn
                 : string -> instream
val closeIn
                 : instream -> unit
val input
                : instream -> vector
val inputAll
                : instream -> vector
val inputNoBlock : instream -> vector option
val input1 : instream -> elem option
                 : instream * int -> vector
val inputN
                 : instream -> string
val inputLine
val endOfStream : instream -> bool
                 : instream -> elem option
val lookahead
type cs character source state
                : ((char, cs) StringCvt.reader -> ('a, cs) StringCvt.reader)
val scanStream
                   -> instream -> 'a option
val stdIn
                 : instream
Text output
type outstream
val openOut
                 : string -> outstream
val openAppend : string -> outstream
                : outstream -> unit
val closeOut
                 : outstream * vector -> unit
val output
               : outstream * elem -> unit
val output1
val outputSubstr : outstream * substring -> unit
val flushOut
                : outstream -> unit
val stdOut
                 : outstream
val stdErr
                 : outstream
val print
                : string -> unit
```

This structure provides input/output functions on text streams. The functions are state-based: reading from or writing to a stream changes the state of the stream. The streams are buffered: output to a stream may not immediately affect the underlying file or device.

Note that under DOS, Windows, OS/2, and MacOS, text streams will be 'translated' by converting (e.g.) the double newline CRLF to a single newline character \n .

[instream] is the type of state-based characters input streams.

[outstream] is the type of state-based character output streams.

[elem] is the type char of characters.

[vector] is the type of character vectors, that is, strings.

TEXT INPUT:

[openIn s] creates a new instream associated with the file named s. Raises Io.Io is file s does not exist or is not accessible.

[closeIn istr] closes stream istr. Has no effect if istr is closed

TEXTIO 145

already. Further operations on istr will behave as if istr is at end of stream (that is, will return "" or NONE or true).

[input istr] reads some elements from istr, returning a vector v of those elements. The vector will be empty (size v=0) if and only if istr is at end of stream or is closed. May block (not return until data are available in the external world).

[inputAll istr] reads and returns the string v of all characters remaining in istr up to end of stream.

[inputNoBlock istr] returns SOME(v) if some elements v can be read without blocking; returns SOME("") if it can be determined without blocking that istr is at end of stream; returns NONE otherwise. If istr does not support non-blocking input, raises Io.NonblockingNotSupported.

[input1 istr] returns SOME(e) if at least one element e of istr is available; returns NONE if istr is at end of stream or is closed; blocks if necessary until one of these conditions holds.

[inputN(istr, n)] returns the next n characters from istr as a string, if that many are available; returns all remaining characters if end of stream is reached before n characters are available; blocks if necessary until one of these conditions holds. (This is the behaviour of the 'input' function prescribed in the 1990 Definition of Standard ML).

[inputLine istr] returns one line of text, including the terminating newline character. If end of stream is reached before a newline character, then the remaining part of the stream is returned, with a newline character added. If istr is at end of stream or is closed, then the empty string "" is returned.

[endOfStream istr] returns false if any elements are available in istr; returns true if istr is at end of stream or closed; blocks if necessary until one of these conditions holds.

[lookahead istr] returns SOME(e) where e is the next element in the stream; returns NONE if istr is at end of stream or is closed; blocks if necessary until one of these conditions holds. Does not advance the stream.

[stdIn] is the buffered state-based standard input stream.

[scanStream scan istr] turns the instream istr into a character source and applies the scanner 'scan' to that source. See StringCvt for more on character sources and scanners. The Moscow ML implementation currently can backtrack only 512 characters, and raises Fail if the scanner backtracks further than that.

TEXT OUTPUT:

[openOut s] creates a new outstream associated with the file named s. If file s does not exist, and the directory exists and is writable, then a new file is created. If file s exists, it is truncated (any existing contents are lost).

[openAppend s] creates a new outstream associated with the file named s. If file s does not exist, and the directory exists and is writable, then a new file is created. If file s exists, any existing contents are retained, and output goes at the end of the file.

[closeOut ostr] closes stream ostr; further operations on ostr (except for additional close operations) will raise exception Io.Io.

[output(ostr, v)] writes the string v on outstream ostr.

[output1(ostr, e)] writes the character e on outstream ostr.

146 TEXTIO

[flushOut ostr] flushes the outstream ostr, so that all data written to ostr becomes available to the underlying file or device.

[stdOut] is the buffered state-based standard output stream.

[stdErr] is the unbuffered state-based standard error stream. That is, it is always kept flushed, so flushOut(stdErr) is redundant.

[print s] outputs s to stdOut and flushes immediately.

The functions below are not yet implemented:

[setPosIn(istr, i)] sets istr to the (untranslated) position i. Raises Io.Io if not supported on istr.

[getPosIn istr] returns the (untranslated) current position of istr. Raises Io.Io if not supported on istr.

[endPosIn istr] returns the (untranslated) last position of istr.
Because of translation, one cannot expect to read
 endPosIn istr - getPosIn istr
from the current position.

[getPosOut ostr] returns the current position in stream ostr. Raises Io.Io if not supported on ostr.

[endPosOut ostr] returns the ending position in stream ostr. Raises Io.Io if not supported on ostr.

[setPosOut(ostr, i)] sets the current position in stream to ostr to i. Raises Io.Io if not supported on ostr.

 $[{\tt mkInstream\ sistr}]$ creates a state-based instream from the functional instream sistr.

[getInstream istr] returns the functional instream underlying the state-based instream istr.

[setInstream(istr, sistr)] redirects istr, so that subsequent input is taken from the functional instream sistr.

 $[{\tt mkOutstream\ sostr}]$ creates a state-based outstream from the outstream sostr.

[getOutstream ostr] returns the outstream underlying the state-based outstream ostr. $% \label{eq:continuous}%$

[setOutstream(ostr, sostr)] redirects the outstream ostr so that subsequent output goes to sostr.

TIME 147

Module Time

```
Time -- SML Basis Library
eqtype time
exception Time
val zeroTime
                     : time
                     : unit -> time
val now
val toSeconds
                     : time -> int
                    : time -> int
val toMilliseconds
val toMicroseconds : time -> int
val fromSeconds
                     : int -> time
val fromMilliseconds : int -> time
val fromMicroseconds : int -> time
val fromReal
                     : real -> time
val toReal
                     : time -> real
val toString
                     : time -> string rounded to millisecond precision
                      : int -> time -> string
val fmt
val fromString
                     : string -> time option
                      : (char, 'a) StringCvt.reader
                       -> (time, 'a) StringCvt.reader
val +
           : time * time -> time
           : time * time -> time
val -
val <
            : time * time -> bool
           : time * time -> bool
val <=
           : time * time -> bool
val >
            : time * time -> bool
val compare : time * time -> order
```

[time] is a type for representing durations as well as absolute points in time (which can be thought of as durations since some fixed time zero). Times can be negative, zero, or positive.

[zeroTime] represents the 0-second duration, and the origin of time, so zeroTime + t = t + zeroTime = t for all t.

[now ()] returns the point in time at which the application occurs.

[fromSeconds s] returns the time value corresponding to s seconds.

[fromMilliseconds ms] returns the time value corresponding to ms milliseconds.

[fromMicroseconds us] returns the time value corresponding to us microseconds.

[toSeconds t] returns the number of seconds represented by t, truncated (towards zero). Raises Overflow if that number is not representable as an int.

[toMilliseconds t] returns the number of milliseconds represented by t, truncated (towards zero). Raises Overflow if that number is not representable as an int.

[toMicroseconds t] returns the number of microseconds represented by t, truncated (towards zero). Raises Overflow if t that number is not representable as an int.

[fromReal r] converts a real to a time value representing that many seconds. It holds that fromReal 0.0 = zeroTime.

[toReal t] converts a time to the number of seconds it represents; hence fromReal and toReal are inverses of each other.

148 TIME

[fmt n t] returns as a string the number of seconds represented by t, rounded to n decimal digits. If n \ll 0, then no decimal digits are reported.

[toString t] returns as a string the number of seconds represented by t, rounded to 3 decimal digits. Equivalent to (fmt 3 t).

[fromString s] returns SOME t where t is the time value represented by the string s of form [\n\t]*[+~-]?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+)); or returns NONE if s cannot be parsed as a time value.

[scan getc src], where getc is a character accessor, returns SOME (t, rest) where t is a time and rest is rest of the input, or NONE if s cannot be parsed as a time value.

- [+] adds two time values. For reals r1, r2 >= 0.0, it holds that fromReal r1 + fromReal r2 = fromReal(Real.+(r1,r2)). Raises Overflow if the result is not representable as a time value.
- [-] subtracts a time value from another. That is, t1 t2 is the duration from t2 to t1 (which may be negative). It holds that t zeroTime = t.

[<] [<=] [>] [>] [>=] compares time values. For instance, for reals r1, r2 >= 0.0 it holds that fromReal r1 < fromReal r2 iff Real.<(r1, r2)

[compare(t1, t2)] returns LESS, EQUAL, or GREATER, according as t1 precedes, equals, or follows t2 in time.

TIMER 149

Module Timer

```
Timer -- SML Basis Library

type cpu_timer
type real_timer

val startCPUTimer : unit -> cpu_timer
val totalCPUTimer : unit -> cpu_timer
val checkCPUTime : cpu_timer -> { usr : Time.time, sys : Time.time }

val checkGCTime : cpu_timer -> Time.time

val startRealTimer : unit -> real_timer
val totalRealTimer : unit -> real_timer
val checkRealTime : real_timer -> Time.time
```

[cpu_timer] is the type of timers for measuring CPU time consumption (user time, garbage collection time, and system time).

[real_timer] is the type of timers for measuring the passing of real time (wall-clock time).

[startCPUTimer ()] returns a cpu_timer started at the moment of the call.

[totalCPUTimer ()] returns a cpu_timer started at the moment the library was loaded.

[checkCPUTime tmr] returns {usr, sys} where usr is the amount of user CPU time consumed since tmr was started and sys is the amount of system CPU time consumed since tmr was started. Note that garbage collection time is included in the usr time. Under MS DOS and MS Windows, usr time is measured as real time.

[checkGCTime tmr] returns the amount of user CPU time spent on garbage collection since tmr was started. Under MS DOS and MS Windows, gc time is measured in real time.

[startRealTimer ()] returns a real_timer started at the moment of the call.

[totalRealTimer ()] returns a real_timer started at the moment the library was loaded.

[checkRealTime \mbox{tmr}] returns the amount of real time that has passed since \mbox{tmr} was started.

150 UNIX

Module Unix

This structure allows Moscow ML programs to start other processes and to communicate with them.

Child processes are not automatically terminated when the parent (ML) process terminates. To forcibly terminate a child process pr, use Unix.kill(pr, Signal.term). Then, to remove the terminated process from the operating system tables, call Unix.reap(pr).

The protocol for communication between the ML program and its child process must be designed with some care, typically using non-blocking input for reading from the child process.

[proc] is the type of processes started by the ML program.

[signal] is the type of Unix-style signals, which can be sent to another process. Signal values must be obtained from the Signal structure.

[execute (cmd, args)] asks the operating system to execute the command cmd with the argument list args, as a separate process. Two pipes connected to the standard input and standard output of the new process are created; these may be obtained using streamsOf. A proc value representing the new process is returned. The new process executes using the same environment as the calling process. Raises Fail in case of failure, e.g. if the process or the pipes cannot be created.

Typically, the cmd argument will be the full pathname of an executable. On Unix systems, simple command searching as done by the shell, allowing cmd to be a relative pathname, can be achieved by using

```
execute("/bin/sh", "-c" :: concat (cmd :: " " :: args))
```

[executeInEnv (cmd, args, env)] asks the operating system to execute the command cmd with the argument list args in the environment env, as a separate process. Returns a proc value representing the new process. Typically, a string in the env list has the form "NAME=VALUE". See also Process.getEnv.

[streamsOf pr] returns a pair (ins, outs) of input and output streams associated with process pr. The standard output of pr is the source for the input stream ins, and the standard input of pr is the sink for the output stream outs.

[reap pr] closes the input and output streams associated with pr, and then suspends the current (ML) process until the process corresponding to pr terminates. Returns the exit status given by pr when it terminated. Raises Fail in case of failure, e.g. if pr has already been reaped.

Under Unix, information about a terminated process remains in the system tables until the process is reaped. Thus, an ML program using execute or executeInEnv must make sure to reap any process it has created, or else the system tables will fill up.

[kill (pr, s)] sends the signal s to the process pr. Raises Fail in case of failure, e.g. if pr has already been killed.

VECTOR 151

Module Vector

Vector -- SML Basis Library type 'a vector = 'a vector val maxLen : int val fromList : 'a list -> 'a vector val tabulate : int * (int -> 'a) -> 'a vector val length : 'a vector -> int : 'a vector * int -> 'a : 'a vector * int * 'a -> 'a vector : 'a vector list -> 'a vector val sub val update val concat val find $: ('a \rightarrow bool) \rightarrow 'a \ vector \rightarrow 'a \ option$ val exists $: ('a \rightarrow bool) \rightarrow 'a \ vector \rightarrow bool$: ('a -> bool) -> 'a vector -> bool val all val app : ('a -> unit) -> 'a vector -> unit : ('a -> 'b) -> 'a vector -> 'b vector : ('a * 'b -> 'b) -> 'b -> 'a vector -> 'b val map val foldl : ('a * 'b -> 'b) -> 'b -> 'a vector -> 'b val foldr : (int * 'a -> bool) -> 'a vector -> (int * 'a) option val findi val foldri : (int * 'a * 'b -> 'b) -> 'b -> 'a vector -> 'b val collate : ('a * 'a -> order) -> 'a vector * 'a vector -> order

['ty vector] is the type of one-dimensional, immutable, zero-based constant-time-access vectors with elements of type 'ty.

Type 'ty vector admits equality if 'ty does. Vectors v1 and v2 are equal if they have the same length and their elements are equal.

[maxLen] is the maximal number of elements in a vector.

[fromList xs] returns a vector whose elements are those of xs. Raises Size if length xs > maxLen.

[tabulate(n, f)] returns a vector of length n whose elements are f 0, f 1, ..., f (n-1), created from left to right. Raises Size if n<0 or n>maxLen.

[length v] returns the number of elements in v.

[sub(v, i)] returns the i'th element of v, counting from 0. Raises Subscript if i<0 or i>=length v.

[update(v, i, x)] creates a copy of v, sets position i to x, and returns the new vector. In contrast to Array.update, this is not a constant-time operation, because it must copy the entire vector. Raises Subscript if i<0 or i>=length v.

[concat vs] returns a vector which is the concatenation from left to right og the vectors in vs. Raises Size if the sum of the sizes of the vectors in vs is larger than maxLen.

[find p v] applies p to each element x of v, from left to right, until p(x) evaluates to true; returns SOME x if such an x exists, otherwise NONE.

[exists p v] applies p to each element x of v, from left to right, until p(x) evaluates to true; returns true if such an x exists, otherwise false.

[all p v] applies p to each element x of v, from left to right, until p(x) evaluates to false; returns false if such an x exists, otherwise true.

152 VECTOR

[foldl f e v] folds function f over v from left to right. That is, computes f(v[len-1], f(v[len-2], ..., f(v[1], f(v[0], e)) ...)), where len is the length of v. [foldr f e v] folds function f over v from right to left. That is, computes f(v[0], f(v[1], ..., f(v[len-2], f(v[len-1], e)) ...)), where len is the length of v. [app f v] applies f to v[j] for j=0,1,..., length v-1. [map f v] applies f to v[j] for j=0,1,...,length v-1 and returns a new vector containing the results. The following iterators generalize the above ones by passing also the vector element index j to the function being iterated. [findi p a] applies f to successive pairs (j, a[j]) for j=0,1,...,n-1, until p(j, a[j]) evaluates to true; returns SOME (j, a[j]) if such a pair exists, otherwise NONE. [foldli f e v] folds function f over the vector from left to right. That is, computes f(n-1, v[n-1], f(..., f(1, v[1], f(0, v[0], e)) ...))where n = length v. [foldri f e v] folds function f over the vector from right to left. That is, computes f(0, v[0], f(1, v[1], ..., f(n-1, v[n-1], e) ...))where n = length v. [appi f v] applies f to successive pairs (j, v[j]) for j=0,1,...,n-1 where n = length v. [mapi f v] applies f to successive pairs (j, v[j]) for $j=0,1,\ldots,n-1$ where $n=length\ v$ and returns a new vector containing the results.

[collate cmp (xs, ys)] returns LESS, EQUAL or GREATER according as xs precedes, equals or follows ys in the lexicographic ordering on vectors induced by the ordering cmp on elements.

VECTORSLICE 153

Module VectorSlice

```
VectorSlice -- SML Basis Library
type 'a slice
val length
             : 'a slice -> int
              : 'a slice * int -> 'a
               : 'a Vector.vector * int * int option -> 'a slice
val slice
              : 'a Vector.vector -> 'a slice
val full
val subslice : 'a slice * int * int option -> 'a slice
             : 'a slice -> 'a Vector.vector * int * int
: 'a slice -> 'a Vector.vector
val base
val vector
val concat : 'a slice list -> 'a Vector.vector
val isEmpty : 'a slice -> bool val getItem : 'a slice -> ('a * 'a slice) option
              : ('a -> bool) -> 'a slice -> 'a option
val exists : ('a -> bool) -> 'a slice -> bool val all : ('a -> bool) -> 'a slice -> bool
val app
              : ('a -> unit) -> 'a slice -> unit
             : ('a -> 'b) -> 'a slice -> 'b Vector.vector

: ('a * 'b -> 'b) -> 'b -> 'a slice -> 'b

: ('a * 'b -> 'b) -> 'b -> 'a slice -> 'b
val map
val foldl
val foldr
               : (int * 'a -> bool) -> 'a slice -> (int * 'a) option
val findi
val appi
              : (int * 'a -> unit) -> 'a slice -> unit
              : (int * 'a -> 'b) -> 'a slice -> 'b Vector.vector
val mapi
              : (int * 'a * 'b -> 'b) -> 'b -> 'a slice -> 'b
val foldli
val foldri : (int * 'a * 'b -> 'b) -> 'a slice -> 'b
val collate : ('a * 'a -> order) -> 'a slice * 'a slice -> order
   ['ty slice] is the type of vector slices, that is, sub-vectors.
```

[slice (a, i, SOME n)] creates the slice (a, i, n), consisting of the sub-vector of a with length n starting at i. Raises Subscript if i<0 or n<0 or i+n > Vector.length a.

```
slice meaning

(a, 0, NONE) the whole vector a[0..len-1]
(a, 0, SOME n) a left sub-vector (prefix) a[0..n-1]
(a, i, NONE) a right sub-vector (suffix) a[i..len-1]
(a, i, SOME n) a general slice a[i..i+n-1]

[full a] creates the slice (a, 0, Vector.length a).

Equivalent to slice(a,0,NONE)

[subslice (sli, i', NONE)] returns the slice (a, i+i', n-i') when sli = (a,i,n). Raises Subscript if i' < 0 or i' > n.

[subslice (sli, i', SOME n')] returns the slice (a, i+i', n') when sli = (a,i,n). Raises Subscript if i' < 0 or n' < 0 or i'+n' > n.
```

154 VECTORSLICE

[base sli] is the concrete triple (a, i, n) when sli = (a, i, n).

[vector sli] creates and returns a vector consisting of the elements of the slice, that is, a[i..i+n-1] when sli = (a,i,n).

[concat slis] creates a vector containing the concatenation of the slices in slis.

[isEmpty sli] returns true if the slice sli = (a,i,n) is empty, that is, if n=0.

[getItem sli] returns SOME(x, rst) where x is the first element and rst the remainder of sli, if sli is non-empty; otherwise returns NONE

[find p sli] applies p to each element x of sli, from left to right, until p(x) evaluates to true; returns SOME x if such an x exists, otherwise NONE.

[exists p sli] applies p to each element x of sli, from left to right, until p(x) evaluates to true; returns true if such an x exists, otherwise false.

[all p sli] applies p to each element x of sli, from left to right, until p(x) evaluates to false; returns false if such an x exists, otherwise true.

[app f sli] applies f to all elements of sli = (a,i,n), from left to right. That is, applies f to a[j+i] for j=0,1,...,n.

[map f sli] applies f to all elements of sli = (a,i,n), from left to right, and returns a vector of the results.

[foldl f e sli] folds function f over sli = (a,i,n) from left to right. That is, computes f(a[i+n-1], f(a[i+n-2],..., f(a[i+1], f(a[i], e))...)).

[foldr f e sli] folds function f over sli = (a,i,n) from right to left. That is, computes f(a[i], f(a[i+1],..., f(a[i+n-2], f(a[i+n-1], e))...)).

The following iterators generalize the above ones by also passing the index into the vector a underlying the slice to the function being iterated.

[findi p sli] applies p to the elements of sli = (a,i,n) and the underlying vector indices, and returns the least (j, a[j]) for which p(j, a[j]) evaluates to true, if any; otherwise returns NONE. That is, evaluates p(j, a[j]) for j=i,..i+n-1 until it evaluates to true for some j, then returns SOME(j, a[j]); otherwise returns NONE.

[appi f sli] applies f to the slice sli = (a,i,n) and the underlying vector indices. That is, applies f to successive pairs (j, a[j]) for j=i,i+1,...,i+n-1.

[mapi f sli] applies f to the slice sli = (a,i,n) and the underlying vector indices, and returns a vector of the results. That is, applies f to successive pairs (j, a[j]) for j=i,i+1,...,i+n-1, and returns #[f(i,a[i]), ..., f(i+n-1,a[i+n-1])].

[foldli f e sli] folds function f over the slice sli = (a,i,n) and the underlying vector indices from left to right. That is, computes f(i+n-1), a[i+n-1], a[i+1], a[i

[foldri f e sli] folds function f over the slice sli = (a,i,n) and the underlying vector indices from right to left. That is, computes f(i, a[i], f(i+1, a[i+1], ..., f(i+n-1, a[i+n-1], e) ...)).

[collate cmp (sli1, sli2)] returns LESS, EQUAL or GREATER according as sli1 precedes, equals or follows sli2 in the lexicographic ordering on slices induced by the ordering cmp on elements.

WEAK 155

Module Weak

Weak --- weak pointers and arrays of weak pointers Single weak pointers type 'a weak val weak : 'a -> 'a weak : 'a weak * 'a -> unit : 'a weak -> 'a val set Raises Fail val get val isweak : 'a weak -> bool Arrays of weak pointers prim_EQtype 'a array val maxLen : int Raises Fail and Subscript val isdead : 'a array * int -> bool val length : 'a array -> int val app : ('a -> unit) -> 'a array -> unit
val foldl : ('a * 'b -> 'b) -> 'b -> 'a array -> 'b
val foldr : ('a * 'b -> 'b) -> 'b -> 'a array -> 'b val modify : ('a -> 'a) -> 'a array -> unit val appi : (int * 'a -> unit) -> 'a array * int * int option -> unit val foldli : (int * 'a * 'b -> 'b) -> 'b -> 'a array * int * int option -> 'b val foldri : (int * 'a * 'b -> 'b) -> 'b -> 'a array * int * int option val modifyi : (int * 'a -> 'a) -> 'a array * int * int option -> unit

['a weak] is the type of weak pointers to objects of type 'a. A weak pointer is a pointer that cannot itself keep an object alive. Hence the object pointed to by a weak pointer may be deallocated by the garbage collector if the object is reachable only by weak pointers. In this case, subsequent accesses via the 'get' function will raise Fail "Dangling weak pointer". (We raise an exception instead of returning an option value, because access via a weak pointer to a deallocated object is likely to be a programming error).

Integers, characters, words and booleans will not be deallocated by the garbage collector and will remain reachable forever by a weak pointer. Reals, strings, tuples and other non-nullary constructors may be deallocated by the garbage collector. Compile-time constants, even composite ones, will not be deallocated either.

[weak v] creates and returns a weak pointer to value v.

[get w] returns the value pointed to by weak pointer w, if the value is still alive. Otherwise raises Fail "Dangling weak pointer".

[set(w, v)] makes the weak pointer w point to the value v.

[isweak w] returns true if the value pointed to by w is dead; returns false otherwise. If an object is reported to be dead, it remains dead. However, an object is reported to be live just if it has not yet been deallocated by the garbage collector. The allocation of any new value may activate the garbage collector and cause the object to die. Thus

if not (isweak w) then get w else "blah" will not raise exception Fail, whereas the following might: if not (isweak w) then ([1,2] @ [3,4]; get w) else "blah" because evaluation of the list append may cause w to die.

The value of isweak w is the same as that of

156 WEAK

(get w; false) handle Fail _ => true but evaluating the latter expression may have the side effect of keeping w alive for slightly longer, because a pointer to w is returned by get w.

_ _ -

['a array] is the type of arrays of weak pointers to objects of type 'a.

A value of type 'a Weak.weak (above) is equivalent to, but more efficient than, a one-element 'a Weak.array. On the other hand, an 'a Weak.array is more efficient than an ('a Weak.weak) Array.array.

[array n] creates an array of n weak pointers. Initially, any access to the array will raise Fail.

[sub(a, i)] returns the object pointed to by cell i (counting from 0) of the array a, if it is live. Raises Fail "Dangling weak pointer" if cell i has never been updated or if the object pointed to has been deallocated by the garbage collector. Raises Subscript if i<0 or i>=length a. To make 'sub' infix, use the declaration infix 9 sub

[update(a, i, v)] updates cell i of array a to point (weakly) to the value v. Raises Subscript if i<0 or i>=length a.

[isdead(a, i)] returns true if the object in cell i of array a is dead, and false otherwise. Analogous to isweak; see above.

[length a] returns the number of elements in a.

[maxLen] is the maximal number of elements in an array.

The iterators described below operate on the live elements only. Note that an element a[k] may die in the course of folding f over earlier elements (e.g. a[1] ... a[k-1]). Thus the functions should be used with great care.

[foldl f e a] folds function f over the live elements of a, from left to right.

[foldr f e a] folds function f over the live elements of a, from right to left.

[app f a] applies f to the live elements of a from left to right.

[modify f a] applies f to a[j] and updates a[j] with the result f(a[j]), for each live element a[j], from left to right.

The following iterators generalize the above ones in two ways:

- . the index j is also being passed to the function being iterated; . the iterators work on a slice (subarray) of an array.
- The slice (a, i, SOME n) denotes the subarray a[i..i+n-1]. That is, a[i] is the first element of the slice, and n is the length of the slice. Valid only if 0 <= i <= i+n <= length a.

The slice (a, i, NONE) denotes the subarray a[i..length a-1]. That is, the slice denotes the suffix of the array starting at i. Valid only if 0 <= i <= length a. Equivalent to (a, i, SOME(length a - i)).

slice	meaning	
(a, 0, NONE) (a, 0, SOME n) (a, i, NONE) (a, i, SOME n)	the whole array a left subarray (prefix) a right subarray (suffix) a general slice	a[0len-1] a[0n-1] a[ilen-1] a[ii+n-1]

[foldli f e (a, i, SOME n)] folds function f over the live elements of the subarray a[i..i+n-1] from left to right. Raises Subscript

WEAK 157

if i<0 or n<0 or i+n > length a.

[foldli f e (a, i, NONE)] folds function f over the live elements of the subarray a[i..len-1] from left to right, where len = length a. Raises Subscript if i<0 or i > length a.

[foldri f e (a, i, SOME n)] folds function f over the live elements of the subarray a[i..i+n-1] from right to left. Raises Subscript if i<0 or n<0 or i+n > length a.

[foldri f e (a, i, NONE)] folds function f over the live elements of the subarray a[i..len-1] from right to left, where len = length a. Raises Subscript if i<0 or i > length a.

[appi f (a, i, SOME n)] applies f to successive pairs (j, a[j]) for j=i,i+1,...,i+n-1, provided a[j] is live. Raises Subscript if i<0 or n<0 or i+n > length a.

[appi f (a, i, NONE)] applies f to successive pairs (j, a[j]) for j=i,i+1,...,len-1, where len = length a, provided a[j] is live. Raises Subscript if i<0 or i > length a.

[modifyi f (a, i, SOME n)] applies f to (j, a[j]) and updates a[j] with the result f(j, a[j]) for j=i,i+1,...,i+n-1, provided a[j] is live. Raises Subscript if i<0 or n<0 or i+n > length a.

[modifyi f (a, i, NONE)] applies f to (j, a[j]) and updates a[j] with the result f(j, a[j]) for $j=i,i+1,\ldots,len-1$, provided a[j] is live. Raises Subscript if i<0 or i > length a.

Module Word

```
Word -- SML Basis Library
type word = word
val wordSize
             : int
              : word * word -> word
val orb
              : word * word -> word
val andb
              : word * word -> word
val xorb
              : word -> word
val notb
val ~
              : word -> word
val <<
              : word * word -> word
              : word * word -> word
val >>
              : word * word -> word
val ~>>
              : word * word -> word
val +
val -
              : word * word -> word
val *
              : word * word -> word
              : word * word -> word
val div
              : word * word -> word
val mod
              : word * word -> bool
val >
              : word * word -> bool
val <
              : word * word -> bool
val >=
              : word * word -> bool
val <=
              : word * word -> order
val compare
             : word * word -> word
val min
              : word * word -> word
val max
             : word -> string
val toString
val fromString : string -> word option
val scan
               : StringCvt.radix
               -> (char, 'a) StringCvt.reader -> (word, 'a) StringCvt.reader
               : StringCvt.radix -> word -> string
val fmt
              : word -> int
: word -> int
val toInt
val toIntX
                                       with sign extension
val fromInt
            : int -> word
val toLargeWord
                : word -> word
val toLargeWordX : word -> word
                                       with sign extension
val fromLargeWord : word -> word
                 : word -> int
val toLargeInt
val toLargeIntX : word -> int
                                       with sign extension
val fromLargeInt : int -> word
   [word] is the type of n-bit words, or n-bit unsigned integers.
   [wordSize] is the value of n above. In Moscow ML, n=31 on 32-bit
  machines and n=63 on 64-bit machines.
   [orb(w1, w2)] returns the bitwise 'or' of w1 and w2.
```

[wordSize] is the value of n above. In Moscow ML, n=31 on 32-bit
machines and n=63 on 64-bit machines.

[orb(w1, w2)] returns the bitwise 'or' of w1 and w2.

[andb(w1, w2)] returns the bitwise 'and' of w1 and w2.

[xorb(w1, w2)] returns the bitwise 'exclusive or' or w1 and w2.

[notb w] returns the bitwise negation (one's complement) of w.

[~ w] returns the arithmetic negation (two's complement) of w.

[<<(w, k)] returns the word resulting from shifting w left by k
bits. The bits shifted in are zero, so this is a logical shift.
Consequently, the result is 0-bits when k >= wordSize.

[>>(w, k)] returns the word resulting from shifting w right by k

bits. The bits shifted in are zero, so this is a logical shift. Consequently, the result is 0-bits when $k \ge wordSize$.

[~>>(w, k)] returns the word resulting from shifting w right by k bits. The bits shifted in are replications of the left-most bit: the 'sign bit', so this is an arithmetical shift. Consequently, for k >= wordSize and wordToInt w >= 0 the result is all 0-bits, and for k >= wordSize and wordToInt w < 0 the result is all 1-bits.

To make <<, >>, and ~>> infix, use the declaration infix 5 << >> ~>>

[+]

[*] [div]

[mod] represent unsigned integer addition, subtraction, multiplication, division, and remainder, modulus 2 raised to the n'th power, where n=wordSize. The operations (i div j) and (i mod j) raise Div when j=0. Otherwise no exceptions are raised.

[<] [<=] [>]

[>=] compare words as unsigned integers.

[compare(w1, w2)] returns LESS, EQUAL, or GREATER, according
as w1 is less than, equal to, or greater than w2 (as unsigned integers).

[min(w1, w2)] returns the smaller of w1 and w2 (as unsigned integers).

[max(w1, w2)] returns the larger of w1 and w2 (as unsigned integers).

[fmt radix w] returns a string representing w, in the radix (base) specified by radix.

radix	description		output format
BIN OCT DEC	unsigned binary unsigned octal unsigned decimal	(base 2) (base 8) (base 10)	[01]+ [0-7]+ [0-9]+
HEX	unsigned hexadecimal	(base 16)	[0-9A-F]+

[toString w] returns a string representing w in unsigned hexadecimal format. Equivalent to (fmt HEX w).

[fromString s] returns SOME(w) if a hexadecimal unsigned numeral can be scanned from a prefix of string s, ignoring any initial whitespace; returns NONE otherwise. Raises Overflow if the scanned number cannot be represented as a word. An unsigned hexadecimal numeral must have form, after possible initial whitespace: [0-9a-fA-F]+

[scan radix getc charsrc] attempts to scan an unsigned numeral from the character source charsrc, using the accessor getc, and ignoring any initial whitespace. The radix argument specifies the base of the numeral (BIN, OCT, DEC, HEX). If successful, it returns SOME(w, rest) where w is the value of the numeral scanned, and rest is the unused part of the character source. Raises Overflow if the scanned number cannot be represented as a word. A numeral must have form, after possible initial whitespace:

radix	input format
BIN OCT	(0w)?[0-1]+ (0w)?[0-7]+
DEC	(0w)?[0-9]+
HEX	(0wx 0wX 0x 0X)?[0-9a-fA-F]+

[toInt w] returns the (signed) integer represented by bit-pattern w. [toIntX w] returns the (signed) integer represented by bit-pattern w. [fromInt i] returns the word representing integer i.

[toLargeInt w] returns the (signed) integer represented by bit-pattern w. [toLargeIntX w] returns the (signed) integer represented by bit-pattern w. [fromLargeInt i] returns the word representing integer i.

[toLargeWord w] returns w.
[toLargeWordX w] returns w.
[fromLargeWord w] returns w.

WORD8 161

Module Word8

```
Word8 -- SML Basis Library
type word = word8
val wordSize : int
              : word * word -> word
val orb
              : word * word -> word
val andb
             : word * word -> word
val xorb
             : word -> word
val notb
val ~
              : word -> word
val <<
              : word * Word.word -> word
              : word * Word.word -> word
val >>
              : word * Word.word -> word
val ~>>
              : word * word -> word
val +
              : word * word -> word
val -
val *
             : word * word -> word
             : word * word -> word
val div
              : word * word -> word
val mod
              : word * word -> bool
              : word * word -> bool
val <
              : word * word -> bool
val >=
             : word * word -> bool
val <=
val compare : word * word -> order
             : word * word -> word
val min
              : word * word -> word
val max
val toString : word -> string
val fromString : string -> word option
              : StringCvt.radix
val scan
              -> (char, 'a) StringCvt.reader -> (word, 'a) StringCvt.reader
val fmt
              : StringCvt.radix -> word -> string
              : word -> int
: word -> int
val toInt
val toIntX
                                       with sign extension
val fromInt
            : int -> word
val toLargeInt
                : word -> int
val toLargeIntX : word -> int
                                       with sign extension
val fromLargeInt : int -> word
val toLargeWord : word -> Word.word
val toLargeWordX : word -> Word.word
                                       with sign extension
val fromLargeWord : Word.word -> word
   [word] is the type of 8-bit words, or 8-bit unsigned integers in
   the range 0..255.
   [wordSize] equals 8.
   [orb(w1, w2)] returns the bitwise 'or' of w1 and w2.
   [andb(w1, w2)] returns the bitwise 'and' of w1 and w2.
   [xorb(w1, w2)] returns the bitwise 'exclusive or' or w1 and w2.
   [notb w] returns the bitwise negation (one's complement) of w.
   [~ w] returns the arithmetic negation (two's complement) of w.
   [<<(w, k)] returns the word resulting from shifting w left by k
   bits. The bits shifted in are zero, so this is a logical shift.
   Consequently, the result is 0-bits when k \ge wordSize.
```

[>>(w, k)] returns the word resulting from shifting w right by k

bits. The bits shifted in are zero, so this is a logical shift. Consequently, the result is 0-bits when $k \ge wordSize$.

[~>>(w, k)] returns the word resulting from shifting w right by k bits. The bits shifted in are replications of the left-most bit: the 'sign bit', so this is an arithmetical shift. Consequently, for k >= wordSize and wordToInt w >= 0 the result is all 0-bits, and for k >= wordSize and wordToInt w < 0 the result is all 1-bits.

To make <<, >>, and ~>> infix, use the declaration: infix 5 << >> ~>>

[+] [-] [*]

[*] [div]

[mod] represent unsigned integer addition, subtraction, multiplication, division, and remainder, modulus 256. The operations (i div j) and (i mod j) raise Div when j = 0. Otherwise no exceptions are raised.

[<] [<=]

[>=] compare words as unsigned integers.

[compare(w1, w2)] returns LESS, EQUAL, or GREATER, according
as w1 is less than, equal to, or greater than w2 (as unsigned integers).

[min(w1, w2)] returns the smaller of w1 and w2 (as unsigned integers).

[max(w1, w2)] returns the larger of w1 and w2 (as unsigned integers).

[fmt radix w] returns a string representing w, in the radix (base) specified by radix.

radix	description		output format
BIN	unsigned binary	(base 2)	[01]+
OCT	unsigned octal	(base 8)	[0-7]+
DEC	unsigned decimal	(base 10)	[0-9]+
HEX	unsigned hexadecimal	(base 16)	[0-9A-F]+

[toString w] returns a string representing w in unsigned hexadecimal format. Equivalent to (fmt HEX w).

[fromString s] returns SOME(w) if a hexadecimal unsigned numeral can be scanned from a prefix of string s, ignoring any initial whitespace; returns NONE otherwise. Raises Overflow if the scanned number cannot be represented as a word. An unsigned hexadecimal numeral must have form, after possible initial whitespace: [0-9a-fA-F]+

[scan radix {getc} charsrc] attempts to scan an unsigned numeral from the character source charsrc, using the accessor getc, and ignoring any initial whitespace. The radix argument specifies the base of the numeral (BIN, OCT, DEC, HEX). If successful, it returns SOME(w, rest) where w is the value of the numeral scanned, and rest is the unused part of the character source. Raises Overflow if the scanned number cannot be represented as a word. A numeral must have form, after possible initial whitespace:

radix	input format
BIN OCT	(0w)?[0-1]+ (0w)?[0-7]+
DEC	(0w)?[0-9]+
HEX	(0wx 0wX 0x 0X)?[0-9a-fA-F]+

[toInt w] returns the integer in the range 0..255 represented by w.

[toIntX w] returns the signed integer (in the range ~128..127)

WORD8 163

represented by bit-pattern w.

[fromInt i] returns the word holding the 8 least significant bits of i.

[toLargeInt w] returns the integer in the range 0..255 represented by w.

[toLargeIntX w] returns the signed integer (in the range $\sim 128..127$) represented by bit-pattern w.

[fromLargeInt i] returns the word holding the 8 least significant bits of i.

[toLargeWord w] returns the Word.word value corresponding to w.

[toLargeWordX w] returns the Word.word value corresponding to w, with sign extension. That is, the 8 least significant bits of the result are those of w, and the remaining bits are all equal to the most significant bit of w: its 'sign bit'.

[fromLargeWord w] returns w modulo 256.

164 WORD8ARRAY

Module Word8Array

```
Word8Array -- SML Basis Library
eqtype array
type elem = Word8.word
type vector = Word8Vector.vector
val maxLen : int
           : int * elem -> array
val array
val tabulate : int * (int -> elem) -> array
val fromList : elem list -> array
            : array -> int
val length
            : array * int -> elem
: array * int * elem -> unit
val sub
val update
            : array -> vector
val vector
            : {src: array, dst: array, di: int} -> unit
val copyVec : {src: vector, dst: array, di: int} -> unit
: (elem -> bool) -> array -> bool
val all
             : (elem -> unit) -> array -> unit
val foldl
            : (elem * 'b -> 'b) -> 'b -> array -> 'b
            : (elem * 'b -> 'b) -> 'b -> array -> 'b
: (elem -> elem) -> array -> unit
val foldr
val modify
val findi
            : (int * elem -> bool) -> array -> (int * elem) option
            : (int * elem -> unit) -> array -> unit
: (int * elem * 'b -> 'b) -> 'b -> array -> 'b
val appi
val foldli
val foldri : (int * elem * 'b -> 'b) -> 'b -> array -> 'b
val modifyi : (int * elem -> elem) -> array -> unit
val collate : (elem * elem -> order) -> array * array -> order
```

[array] is the type of one-dimensional, mutable, zero-based constant-time-access arrays with elements of type Word8.word, that is, 8-bit words. Arrays al and a2 are equal if both were created by the same call to a primitive (array0, array, tabulate, fromList).

All operations are as for Array.array.

WORD8ARRAYSLICE 165

Module Word8ArraySlice

```
Word8ArraySlice -- SML Basis Library
type elem = Word8.word
type array = Word8Array.array
type vector = Word8Vector.vector
type vector_slice = Word8VectorSlice.slice
type slice
               : slice -> int
val length
               : slice * int -> elem
: slice * int * elem -> unit
val sub
val update
                : array * int * int option -> slice
val slice
val full : array -> slice
val subslice : slice * int * int option -> slice
               : slice -> array * int * int
                : slice -> vector
val vector
                : {src: slice, dst: array, di: int} -> unit
val copy
              : {src: vector_slice, dst: array, di: int} -> unit
val copyVec
val isEmpty : slice -> bool
val getItem : slice -> (elem * slice) option
val find
                : (elem -> bool) -> slice -> elem option
               : (elem -> bool) -> slice -> bool
: (elem -> bool) -> slice -> bool
val exists
val all
               : (elem -> unit) -> slice -> unit
: (elem * 'b -> 'b) -> 'b -> slice -> 'b
: (elem * 'b -> 'b) -> 'b -> slice -> 'b
val app
val foldl
val foldr
               : (elem -> elem) -> slice -> unit
val modify
                : (int * elem -> bool) -> slice -> (int * elem) option
val findi
               : (int * elem -> bool/ -> slice -> (int * elem : (int * elem * 'b -> 'b) -> 'b -> slice -> 'b : (int * elem * 'b -> 'b) -> 'b -> slice -> 'b
val appi
val foldli
val foldri
val modifyi : (int * elem -> elem) -> slice -> unit
val collate : (elem * elem -> order) -> slice * slice -> order
```

```
[slice] is the type of Word8Array slices, that is, sub-arrays of Word8Array.array values. The slice (a,i,n) is valid if 0 <= i <= i+n <= size s, or equivalently, 0 <= i and 0 <= n and i+n <= size s. A valid slice sli = (a,i,n) represents the sub-array a[i...i+n-1], so the elements of sli are a[i], a[i+1], ..., a[i+n-1], and n is the length of the slice. Only valid slices can be constructed by the functions below.
```

All operations are as for ArraySlice.slice.

166 WORD8VECTOR

Module Word8Vector

```
Word8Vector -- SML Basis Library
eqtype vector
type elem = Word8.word
val maxLen : int
val fromList : elem list -> vector
val tabulate : int * (int -> elem) -> vector
val length
          : vector -> int
           : vector * int -> elem
val sub
          : vector * int * elem -> vector
val update
          : vector list -> vector
val concat
           : (elem -> bool) -> vector -> elem option
val app
           : (elem -> unit) -> vector -> unit
          : (elem -> elem) -> vector -> vector
: (elem * 'b -> 'b) -> 'b -> vector -> 'b
val map
val foldl
          : (elem * 'b -> 'b) -> 'b -> vector -> 'b
val foldr
           : (int * elem -> bool) -> vector -> (int * elem) option
val findi
           : (int * elem -> unit) -> vector -> unit
val appi
val collate : (elem * elem -> order) -> vector * vector -> order
```

[vector] is the type of one-dimensional, immutable, zero-based constant-time-access vectors with elements of type Word8.word, that is, 8-bit words. Type vector admits equality, and vectors v1 and v2 are equal if they have the same length and their elements are equal.

All operations are as for Vector.vector.

WORD8VECTORSLICE 167

Module Word8VectorSlice

```
Word8VectorSlice -- SML Basis Library
type elem = Word8.word
type vector = Word8Vector.vector
type slice
               : slice -> int
val length
                : slice * int -> elem
val sub
val slice
               : vector * int * int option -> slice
val full
                : vector -> slice
val subslice : slice * int * int option -> slice
              : slice -> vector * int * int : slice -> vector
val base
val vector
               : slice list -> vector
val concat
val isEmpty : slice -> bool
val getItem : slice -> (elem * slice) option
               : (elem -> bool) -> slice -> elem option
val find
val exists : (elem -> bool) -> slice -> bool
val all
                : (elem -> bool) -> slice -> bool
val app
                : (elem -> unit) -> slice -> unit
               : (elem -> elem) -> slice -> vector
: (elem * 'b -> 'b) -> 'b -> slice -> 'b
val map
val foldl
               : (elem * 'b -> 'b) -> 'b -> slice -> 'b
val foldr
               : (int * elem -> bool) -> slice -> (int * elem) option
: (int * elem -> unit) -> slice -> unit
val findi
val appi
               : (int * elem -> elem) -> slice -> unit

: (int * elem -> elem) -> slice -> vector

: (int * elem * 'b -> 'b) -> 'b -> slice -> 'b

: (int * elem * 'b -> 'b) -> 'b -> slice -> 'b
val mapi
val foldli
val foldri
val collate : (elem * elem -> order) -> slice * slice -> order
```

All operations are as for VectorSlice.slice.

Index

!		<<	
•	value (General), 46, 48		value (Word), 158
! =	value (deficial), 10, 10		value (Word8), 161
: -	volue (Deel) 114 115		value (words), 101
4	value (Real), 114, 115	<=	
\$	() 00		value (Char), 24, 26
	constructor (Msp), 80		value (General), 47, 49
\$\$			value (Int), 53, 54
	constructor (Msp), 80		value (Real), 114
%			value (String), 134, 135
	value (Msp), 78, 80		value (Time), 147, 148
% #			value (Word), 158, 159
	value (Msp), 78, 80		value (Word8), 161, 162
% %	· · · · · · · · · · · · · · · · · · ·	<>	
	value (Msp), 78, 80		value (General), 46, 49
% % #	(1.6), 70, 00	=	(and (and and and and and and and and and and
σоπ	value (Msp), 78, 80		value (General), 46, 49
%?	value (PSP), 70, 60		value (General), 40, 47
6:	value (Mars) 79 90	==	volvo (Dool) 114 115
	value (Msp), 78, 80		value (Real), 114, 115
&&	() 00	>	1 () 24 25
	constructor (Msp), 80		value (Char), 24, 26
*			value (General), 47, 49
	value (General), 47, 49		value (Int), 53, 54
	value (Int), 53		value (Real), 114
	value (Real), 114		value (String), 134, 135
	value (Word), 158, 159		value (Time), 147, 148
	value (Word8), 161, 162		value (Word), 158, 159
+	·		value (Word8), 161, 162
	value (General), 47, 49	>=	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
	value (Int), 53		value (Char), 24, 26
	value (Real), 114		value (General), 47, 49
	value (Time), 147, 148		value (Int), 53, 54
			* * * * * * * * * * * * * * * * * * * *
	value (Word), 158, 159		value (Real), 114
	value (Word8), 161, 162		value (String), 134, 136
-	1 (7 10		value (Time), 147, 148
	value (General), 47, 49		value (Word), 158, 159
	value (Int), 53		value (Word8), 161, 162
	value (Real), 114	>>	
	value (Time), 147, 148		value (Word), 158
	value (Word), 158, 159		value (Word8), 161
	value (Word8), 161, 162	?=	
/			value (Real), 114, 115
	value (General), 47, 49	@	
	value (Real), 114		value (List), 60
:=	` ''	^	, ,,
	value (General), 46, 48		value (General), 46, 49
<	value (delicear), 10, 10		value (String), 134
	value (Char), 24, 26	~	varue (Berring), 134
			value (General), 47, 49
	value (General), 47, 49		
	value (Int), 53, 54		value (Int), 53
	value (Real), 114		value (Real), 114
	value (String), 134, 135		value (Word), 158
	value (Time), 147, 148		value (Word8), 161
	value (Word), 158, 159	~>>	
	value (Word8), 161, 162		value (Word), 158, 159

value (Word8), 161, 162	value (Msp), 79, 81
A_EXEC	ahrefa
constructor (FileSys), 39	value (Msp), 79, 81
A_READ	all value (Array), 3, 4
constructor (FileSys), 39	value (ArraySlice), 8, 9
A_WRITE	value (CharArray), 27
constructor (FileSys), 39	value (CharArraySlice), 28
abrt	value (CharVector), 29
value (Signal), 124	value (CharVectorSlice), 30
Abs	value (Hashset), 50
exception (SML90), 123	value (List), 60, 61
abs	value (ListPair), 62
value (General), 47, 49	value (Substring), 139, 140
value (Int), 53	value (Vector), 151
value (Real), 114	value (VectorSlice), 153, 154
accept	value (Word8Array), 164
value (Socket), 126, 127	value (Word8ArraySlice), 165
access	value (Word8Vector), 166
type (FileSys), 38, 39	value (Word8VectorSlice), 167
value (FileSys), 38, 39	allCookies
acos value (Math), 67	value (Mosmlcookie), 77
active	allEq
type (Socket), 126, 127	value (ListPair), 62, 63
add	AlreadyThere
value (Binaryset), 16	exception (Gdbm), 41
value (Gdbm), 41, 42	exception (Polygdbm), 101
value (Hashset), 50	alrm
value (Intset), 56	value (Signal), 124
value (Polygdbm), 101	aname
value (Rbset), 111	value (Msp), 79, 81
value (Redblackset), 117	andb
value (Splayset), 132	value (Word), 158
add'	value (Word8), 161
value (Rbset), 111	app value (Array), 3, 4
add_break	value (Array2), 5, 6
value (PP), 93	value (ArraySlice), 8, 9
add_newline	value (Binarymap), 15
value (PP), 93, 94	value (Binaryset), 16, 17
add_string	value (CharArray), 27
value (PP), 93, 94 addChar	value (CharArraySlice), 28
value (Buffer), 19	value (CharVector), 29
addList	value (CharVectorSlice), 30
value (Binaryset), 16	value (Gdbm), 41, 42
value (Hashset), 50	value (Hashset), 50, 51
value (Intset), 56	value (Intmap), 55
value (Rbset), 111	value (Intset), 56
value (Redblackset), 117	value (List), 60, 61
value (Splayset), 132	value (ListPair), 62
address	value (NJ93), 88
value (Msp), 79, 81	value (Option), 92
addString	value (Polygdbm), 101, 102
value (Buffer), 19	value (Rbset), 111, 112
addSubString	value (Redblackmap), 116
value (Buffer), 19	value (Redblackset), 117, 118
ahref	value (Regex), 119, 121

value (Splaymap), 131	argv
value (Splayset), 132, 133	value (Mosml), 73
value (Substring), 139, 142	Array (structure), 3–4
value (Vector), 151, 152	array
value (VectorSlice), 153, 154	type (Array), 3
value (Weak), 155, 156	type (Array2), 5
value (Word8Array), 164	type (CharArray), 27
value (Word8ArraySlice), 165	type (CharArraySlice), 28
value (Word8Vector), 166	type (Dynarray), 35
value (Word8VectorSlice), 167	type (Weak), 155, 156
app1	type (Word8Array), 164
value (Callback), 21, 22	type (Word8ArraySlice), 165
value (Dynlib), 36, 37	value (Array), 3
	value (Array2), 5
app2	value (CharArray), 27
value (Callback), 21, 22	value (Dynarray), 35
value (Dynlib), 36, 37	value (Weak), 155, 156
app3	value (Word8Array), 164
value (Callback), 21, 22	Array2 (structure), 5–7
value (Dynlib), 36, 37	
app4	ArraySlice (structure), 8–10
value (Callback), 21, 22	Arraysort (structure), 11
value (Dynlib), 36, 37	as_compile
app5	value (AppleScript), 2
value (Callback), 21, 23	as_dispose
value (Dynlib), 36, 37	value (AppleScript), 2
appEq	as_run_script
value (ListPair), 62, 63	value (AppleScript), 2
appi	as_run_text
value (Array), 3, 4	value (AppleScript), 2
value (Array2), 5, 7	asin
value (ArraySlice), 8, 9	value (Math), 67
value (CharArray), 27	atan
value (CharArraySlice), 28	value (Math), 67
value (CharVector), 29	atan2
	value (Math), 67
value (CharVectorSlice), 30 value (Vector), 151, 152	atExit
	value (Process), 109
value (VectorSlice), 153, 154	
value (Weak), 155, 157	backtrack
value (Word8Array), 164	value (Lexing), 58
value (Word8ArraySlice), 165	base
value (Word8Vector), 166	value (ArraySlice), 8, 9
value (Word8VectorSlice), 167	value (CharArraySlice), 28
AppleScript (structure), 2	value (CharVectorSlice), 30
AppleScriptErr	value (Path), 98, 100
exception (AppleScript), 2	value (Substring), 139, 140
apply	value (VectorSlice), 153, 154
value (Polyhash), 103	value (Word8ArraySlice), 165
applyto	value (Word8VectorSlice), 167
value (Mysql), 85, 87	before
value (Postgres), 106, 108	value (General), 46, 48
arctan	begin_block
value (NJ93), 88	value (PP), 93, 94
value (SML90), 123	Binarymap (structure), 15
area	Binaryset (structure), 15–17
value (Msp), 79, 82	Bind Sincture), 10–17
arguments	exception (General), 46, 48
value (CommandLine), 31	bind

1 (0 1 1) 106 107	
value (Socket), 126, 127	cgi_annotation_server
BinIO (structure), 12–14	value (Mosmlcgi), 74, 76
blockquote	cgi_api_version
value (Msp), 78, 81	value (Mosmlcgi), 74, 76
blockquotea	cgi_auth_type
value (Msp), 78, 81	value (Mosmlcgi), 74
body	cgi_content_length
value (Msp), 78, 81	value (Mosmlcgi), 74, 76
bodya	cgi_content_type
value (Msp), 78, 81	value (Mosmlcgi), 74
Bool (structure), 18	cgi_document_root
bool	value (Mosmlcgi), 74, 76
type (Bool), 18	, = ,
type (General), 46, 47	cgi_field_integer
bound	value (Mosmlcgi), 74, 75
	cgi_field_string
value (Dynarray), 35	value (Mosmlcgi), 74, 75
br	cgi_field_strings
value (Msp), 78, 81	value (Mosmlcgi), 74, 75
bra	cgi_fieldnames
value (Msp), 78, 81	value (Mosmlcgi), 74, 75
break_style	cgi_gateway_interface
type (PP), 93	value (Mosmlcgi), 74, 76
browser	cgi_http_accept
value (Help), 52	value (Mosmlcgi), 74, 76
bucketSizes	
value (Polyhash), 103, 104	cgi_http_cookie
buf	value (Mosmlcgi), 74, 76
type (Buffer), 19	cgi_http_forwarded
type (Socket), 126, 128	value (Mosmlcgi), 74, 76
buff_input	cgi_http_host
	value (Mosmlcgi), 74, 76
value (Nonstdio), 90	cgi_http_proxy_connection
buff_output	value (Mosmlcgi), 74, 76
value (Nonstdio), 90	cgi_http_referer
Buffer (structure), 19	value (Mosmlcgi), 74, 76
bus	cgi_http_user_agent
value (Signal), 124	value (Mosmlcgi), 74, 76
Byte (structure), 20	cgi_is_subreq
bytesToString	value (Mosmlcgi), 74, 76
value (Byte), 20	
byteToChar	cgi_part
value (Byte), 20	value (Mosmlcgi), 74, 75
	cgi_partnames
Callback (structure), 21–23	value (Mosmlcgi), 74, 75
can_input	cgi_parts
value (NJ93), 89	value (Mosmlcgi), 74, 75
caption	cgi_path_info
value (Msp), 79, 82	value (Mosmlcgi), 74, 76
captiona	cgi_path_translated
value (Msp), 79, 82	value (Mosmlcgi), 74, 76
ceil	cgi_query_string
value (General), 46, 49	value (Mosmlcgi), 74, 76
value (Real), 114, 115	cgi_remote_addr
ceiling	value (Mosmlcgi), 74, 76
value (NJ93), 88	cgi_remote_host
center	value (Mosmlcgi), 74, 76
value (Msp), 78, 81	cgi_remote_ident
cflag	value (Mosmlcgi), 74, 76
type (Regex), 119, 120	cgi_remote_user

1 (2 1 1) 74 76	1 () 02 04
value (Mosmlcgi), 74, 76	value (PP), 93, 94
cgi_request_filename	clearParser
value (Mosmlcgi), 74, 76	value (Parsing), 96, 97
cgi_request_method	close
value (Mosmlcgi), 74, 76	value (Socket), 126, 128
cgi_request_uri	close_in
value (Mosmlcgi), 74, 76	value (NJ93), 88
cgi_script_filename	value (SML90), 123
value (Mosmlcgi), 74, 76	close_out
cgi_script_name	value (NJ93), 88
value (Mosmlcgi), 74, 76	value (SML90), 123
cgi_server_admin	closebase
value (Mosmlcgi), 74, 76	value (Mysql), 84, 85
cgi_server_name	value (Postgres), 105, 106
value (Mosmlcgi), 74, 75	Closed
cgi_server_port	exception (Dynlib), 36
value (Mosmlcgi), 74, 76	exception (Gdbm), 41
cgi_server_protocol	exception (Mysql), 84
value (Mosmlcgi), 74, 76	exception (Polygdbm), 101
cgi_server_software	exception (Postgres), 105
value (Mosmlcgi), 74, 75	closeDir
cgi_the_request	value (FileSys), 38
value (Mosmlcgi), 74, 76	closeIn
Char (structure), 24–26	value (BinIO), 12
char (structure), 24–20	value (TextIO), 12
type (Char), 24	closeOut
type (General), 46, 47	value (BinIO), 12, 13
value (Gdimage), 43, 45	value (TextIO), 144, 145
CharArray (structure), 27	cmdtuples
CharArraySlice (structure), 28	value (Mysql), 84, 86
charsize	value (Postgres), 105, 107
value (Gdimage), 43, 45	collate
charToByte	value (Array), 3, 4
value (Byte), 20	value (ArraySlice), 8, 10
charUp	value (CharArray), 27
value (Gdimage), 43, 45	value (CharArraySlice), 28
CharVector (structure), 29	value (CharVector), 29
CharVectorSlice (structure), 30	value (CharVectorSlice), 30
chDir	value (List), 60, 61
value (FileSys), 38	value (String), 134, 135
checkCPUTime	value (Substring), 139, 140
value (Timer), 149	value (Vector), 151, 152
checkGCTime	value (VectorSlice), 153, 154
value (Timer), 149	value (Word8Array), 164
checkRealTime	value (Word8ArraySlice), 165
value (Timer), 149	value (Word8Vector), 166
chld	value (Word8VectorSlice), 167
value (Signal), 124	ColMajor
Chr	constructor (Array2), 5
exception (General), 46	color
chr	constructor (Gdimage), 44
value (Char), 24	type (Gdimage), 43, 44
value (Char), 24 value (NJ93), 88	value (Gdimage), 43, 44
value (SML90), 123	column
clear	value (Array2), 5, 6
value (Buffer), 19	CommandLine (structure), 31
clear_ppstream	comment

value (Msp), 78, 81	value (ArraySlice), 8, 9
compare	value (CharArray), 27
value (Char), 24, 26	value (CharArraySlice), 28
value (Date), 32, 33	value (Gdimage), 43, 45
value (FileSys), 38, 40	value (Hashset), 50
value (Int), 53, 54	value (Polyhash), 103, 104
value (Rbset), 111, 112	value (Word8Array), 164
value (Real), 114	value (Word8ArraySlice), 165
value (Socket), 126, 129	copyResize
value (String), 134, 135	value (Gdimage), 43, 45
value (Substring), 139, 140	copytablefrom
value (Time), 147, 148	value (Mysql), 85, 87
value (Word), 158, 159	value (Postgres), 106, 108
value (Word8), 161, 162	copytableto
compile	value (Mysql), 85, 87
value (Meta), 69, 70	value (Postgres), 106, 108
compileStructure	copyVec
value (Meta), 69, 70	value (Array), 3, 4
compileToplevel	value (ArraySlice), 8, 9
value (Meta), 69, 70	value (CharArray), 27
compose	value (CharArraySlice), 28
value (Option), 92	value (Word8Array), 164
composePartial	value (Word8ArraySlice), 165
value (Option), 92	cos
concat	value (Math), 67
value (CharVector), 29	value (NJ93), 88
value (CharVectorSlice), 30	value (SML90), 123
value (List), 60, 61	cosh
value (Path), 98, 99	value (Math), 67
value (String), 134	cptr type (Callback), 21, 22
value (Substring), 139, 140	cpu_timer
value (Vector), 151	type (Timer), 149
value (VectorSlice), 153, 154	createLexer
value (Word8Vector), 166	value (Lexing), 58
value (Word8VectorSlice), 167	createLexerString
concatWith	value (Lexing), 58
value (String), 134	CS
value (Substring), 139, 140	type (StringCvt), 137
connect	type (TextIO), 144
value (Socket), 126, 128	currentArc
conservative	value (Path), 98, 99
value (Meta), 69, 70	
CONSISTENT	Date
constructor (PP), 93	exception (Date), 32
value (Signal), 124	Date (structure), 32–34
contains	date
value (Char), 24, 25	type (Date), 32
contents	value (Date), 32
value (Buffer), 19	datum
cookiedata	type (Gdbm), 41 day
type (Mosmlcookie), 77	value (Date), 32, 33
CookieError	db
exception (Mosmlcookie), 77	value (Mysql), 84, 85
copy	value (Postgres), 105, 106
value (Array), 3	dbconn
value (Array2), 5, 7	type (Mysql), 84, 85

(-) 105 106	1 (1) 150 150
type (Postgres), 105, 106	value (Word), 158, 159
dbresult	value (Word8), 161, 162
type (Mysql), 84, 85	divi
type (Postgres), 105, 106	value (Msp), 78, 81
dbresultstatus	divia
type (Mysql), 84	value (Msp), 78, 81
type (Mysq1), 64 type (Postgres), 105	dl
• •	
dd	value (Msp), 79, 82
value (Msp), 79, 82	dla
dec	value (Msp), 79, 82
value (NJ93), 88	dlclose
default	value (Dynlib), 36, 37
value (Dynarray), 35	dlHandle
defaultBrowser	type (Dynlib), 36
	dlopen
value (Help), 52	value (Dynlib), 36, 37
delay	
value (Susp), 143	dlsym
delete	value (Dynlib), 36, 37
value (Binaryset), 16	Domain
value (Hashset), 50	exception (General), 46
value (Intset), 56	doubleVec
value (Rbset), 111, 112	value (Mosml), 73
	drawArc
value (Redblackset), 117	value (Gdimage), 43, 45
value (Splayset), 132	drawLine
deleteCookie	value (Gdimage), 43, 45
value (Mosmlcookie), 77	
depth	drawPixel
value (Rbset), 111	value (Gdimage), 43, 45
dest_ppstream	drawPolygon
value (PP), 93	value (Gdimage), 43, 45
	drawRect
dgram	value (Gdimage), 43, 45
type (Socket), 126, 127	drop
dict	value (List), 60
type (Binarymap), 15	dropl
type (Redblackmap), 116	
type (Splaymap), 131	value (StringCvt), 137
difference	value (Substring), 139, 140
value (Binaryset), 16	dropr
value (Intset), 56	value (Substring), 139, 140
value (Ricset), 30 value (Rbset), 111, 112	dt
	value (Msp), 79, 82
value (Redblackset), 117	dummyAction
value (Splayset), 132	value (Lexing), 58
dimensions	Dynarray (structure), 35
value (Array2), 5, 6	Dynlib (structure), 36–37
dir	dyntype
value (Path), 98, 100	
dirstream	type (Mysql), 85
type (FileSys), 38	type (Postgres), 106
	dynval
displayLines	type (Mysql), 84
value (Help), 52	type (Postgres), 105
Div	dynval2s
exception (General), 46	value (Mysql), 85, 87
exception (Real), 114	value (Postgres), 106, 108
value (General), 48	(1 00 0 5 1 0 0), 100, 100
div	е
value (General), 47, 49	value (Math), 67
value (Int), 53	eflag

type (Regex), 119, 120	executeInEnv
elem	value (Unix), 150
type (BinIO), 12	exists
type (CharArray), 27	value (Array), 3, 4
type (CharArraySlice), 28	value (ArraySlice), 8, 9
type (CharVector), 29	value (CharArray), 27
type (CharVectorSlice), 30	value (CharArraySlice), 28
type (Text10), 144	value (CharVector), 29
	value (CharVectorSlice), 30
type (Word8Array), 164	value (Hashset), 50
type (Word8ArraySlice), 165	value (List), 60, 61
type (Word8Vector), 166	
type (Word8VectorSlice), 167	value (ListPair), 62
em	value (Vector), 151
value (Msp), 79, 82	value (VectorSlice), 153, 154
Empty	value (Word8Array), 164
constructor (Msp), 80	value (Word8ArraySlice), 165
exception (List), 60	value (Word8Vector), 166
empty	value (Word8VectorSlice), 167
value (Binaryset), 16	exit
value (Hashset), 50	value (Process), 109
value (Intmap), 55	exn
value (Intset), 56	type (General), 46, 47
value (Rbset), 111	exnMessage
value (Redblackset), 117	value (General), 46, 49
value (Splayset), 132	exnName
end block	value (General), 46, 48
value (PP), 93, 94	exp
end_of_stream	value (Math), 67
value (NJ93), 88	value (NJ93), 88
value (SML90), 123	value (SML90), 123
endOfStream	explode
	value (NJ93), 88
value (BinIO), 12, 13	value (SML90), 123
value (TextIO), 144, 145	value (String), 134, 135
eqclasses	value (Substring), 139, 140
value (Listsort), 64	ext
equal	value (Path), 98, 100
value (Binaryset), 16	Extended
value (Hashset), 50	constructor (Regex), 120
value (Intset), 56	extract
value (Rbset), 111, 112	value (String), 134
value (Redblackset), 117	value (Substring), 139
value (Splayset), 132	(2 2 3),
errLocation	Fail
value (Location), 65	exception (General), 46
errMsg	failure
value (Location), 65	value (Process), 109
errormessage	fast_really_input
value (Mysql), 84, 86	value (Nonstdio), 90
value (Postgres), 105, 107	fastwrite
errorMsg	value (Gdbm), 41, 42
value (OS), 91	value (Polygdbm), 101, 102
errPrompt	fields
value (Location), 65	value (Regex), 119, 122
execute	value (String), 134, 135
value (Mysql), 84, 86	value (Substring), 139, 142
value (Postgres), 105, 107	file
value (Unix), 150	value (Path), 98, 100
······································	,,,

file_exists	value (CharArraySlice), 28
value (Nonstdio), 90	value (CharVector), 29
file_id	value (CharVectorSlice), 30
type (FileSys), 38, 39	value (Vector), 151, 152
fileAddr	value (VectorSlice), 153, 154
value (Socket), 126, 127	value (Word8Array), 164
fileDgram	value (Word8ArraySlice), 165
value (Socket), 126, 127	value (Word8Vector), 166
fileId	value (Word8VectorSlice), 167
value (FileSys), 38, 40	first
fileSize	
	value (Substring), 139, 140
value (FileSys), 38, 39	flag
fileStream	type (Dynlib), 36, 37
value (Socket), 126, 127	flatten
FileSys (structure), 38–40	value (Msp), 78, 80
fill	floatVec
value (Gdimage), 43, 45	value (Mosml), 73
fillBorder	floor
value (Gdimage), 43, 45	value (General), 46, 49
fillPolygon	value (Real), 114, 115
value (Gdimage), 43, 45	flush_out
fillRect	value (NJ93), 89
value (Gdimage), 43, 45	flush_ppstream
filter	value (PP), 93, 94
value (List), 60, 61	flushOut
value (Option), 92	value (BinIO), 12, 13
value (Polyhash), 103	value (TextIO), 144, 146
find	fmt
value (Array), 3, 4	value (Date), 32, 33
value (ArraySlice), 8, 9	value (Int), 53, 54
value (Binarymap), 15	value (Real), 114, 115
value (Binaryset), 16, 17	value (Time), 147, 148
value (CharArray), 27	value (Word), 158, 159
value (CharArraySlice), 28	value (Word8), 161, 162
value (CharVector), 29	fname
value (CharVectorSlice), 30	value (Mysql), 84, 86
value (Gdbm), 41, 42	value (Postgres), 105, 107
value (Hashset), 50, 51	fnames
value (Intset), 56, 57	value (Mysql), 84, 86
value (List), 60, 61	value (Postgres), 105, 107
value (Polygdbm), 101	fnumber
value (Polyhash), 103	value (Mysql), 84, 86
value (Rbset), 111, 112	value (Postgres), 105, 107
value (Redblackmap), 116	fold
value (Redblackset), 117, 118	value (Array2), 5, 6
value (Splaymap), 131	value (Gdbm), 41, 42
value (Splayset), 132, 133	value (Hashset), 50, 51
value (Vector), 151	value (NJ93), 88
value (VectorSlice), 153, 154	value (Polygdbm), 101, 102
value (Word8Array), 164	value (Regex), 119, 122
value (Word8ArraySlice), 165	foldi
value (Word8Vector), 166	value (Array2), 5, 7
value (Word8VectorSlice), 167	foldl
findi	value (Array), 3, 4
value (Array), 3, 4	value (ArraySlice), 8, 9
value (ArraySlice), 8, 9	value (Binarymap), 15
value (CharArray), 27	value (Binaryset), 16, 17

value (Chanhanna) 27	volvo (Verbougline) 152 154
value (CharArray), 27	value (VectorSlice), 153, 154
value (CharArraySlice), 28	value (Weak), 155, 156
value (CharVector), 29	value (Word8Array), 164
value (CharVectorSlice), 30	value (Word8ArraySlice), 165
value (Intmap), 55	value (Word8Vector), 166
value (Intset), 56	value (Word8VectorSlice), 167
value (List), 60, 61	foldrEq
value (ListPair), 62, 63	value (ListPair), 62, 63
value (Rbset), 111, 112	foldri
value (Redblackmap), 116	value (Array), 3, 4
value (Redblackset), 117, 118	value (ArraySlice), 8, 10
value (Splaymap), 131	value (CharArray), 27
value (Splayset), 132, 133	value (CharArraySlice), 28
value (Substring), 139, 142	value (CharVector), 29
value (Vector), 151, 152	value (CharVectorSlice), 30
value (VectorSlice), 153, 154	value (Vector), 151, 152
value (Weak), 155, 156	value (VectorSlice), 153, 154
value (Word8Array), 164	value (Weak), 155, 157
value (Word8ArraySlice), 165	value (Word8Array), 164
value (Word8Vector), 166	value (Word8ArraySlice), 165
value (Word8VectorSlice), 167	value (Word8Vector), 166
foldlEq value (ListPair), 62, 63	value (Word8VectorSlice), 167 font
foldli	type (Gdimage), 43, 44
value (Array), 3, 4	fonta
value (ArraySlice), 8, 9	value (Msp), 79, 82
value (CharArray), 27	force
value (CharArraySlice), 28	value (Susp), 143
value (CharVector), 29	form
value (CharVectorSlice), 30	value (Msp), 79, 83
value (Vector), 151, 152	forma
value (VectorSlice), 153, 154	value (Msp), 79, 83
value (Weak), 155-157	formattable
value (Word8Array), 164	value (Mysql), 85, 87
value (Word8ArraySlice), 165	value (Postgres), 106, 108
value (Word8Vector), 166	fpe
value (Word8VectorSlice), 167	value (Signal), 124
foldr	frag
value (Array), 3, 4	type (General), 46, 48
value (ArraySlice), 8, 9	frame
value (Binarymap), 15	value (Msp), 80, 83
value (Binaryset), 16, 17	framea
value (CharArray), 27	value (Msp), 80, 83
value (CharArraySlice), 28	frameset
value (CharVector), 29	value (Msp), 80, 83
value (CharVectorSlice), 30	fromCString
value (Intmap), 55	value (Char), 24, 26
value (Intset), 56, 57	value (String), 134, 135
value (List), 60, 61	fromDefault
value (ListPair), 62	value (Real), 114
value (Rbset), 111, 112	fromInt
value (Redblackmap), 116	value (Int), 53
value (Redblackset), 117, 118	value (Real), 114
value (Splaymap), 131	value (Word), 158, 159
value (Splayset), 132, 133	value (Words), 161, 163
value (Sprayset), 132, 133 value (Substring), 139, 142	fromLarge
value (Substitug), 139, 142 value (Vector), 151, 152	value (Int), 53
variac (veccess), 131, 132	varue (IIIC), 55

fromLargeInt	value (Word8ArraySlice), 165
value (Word), 158, 160	value (Word8VectorSlice), 167
value (Word8), 161, 163	fullPath
fromLargeWord	value (FileSys), 38, 39
value (Word), 158, 160	an () 11 12
value (Word8), 161, 163	Gdbm (structure), 41–42
fromList	GdbmError
value (Array), 3	exception (Gdbm), 41
value (Array2), 5	exception (Polygdbm), 101
value (CharArray), 27	Gdimage (structure), 43–45
value (CharVector), 29	General (structure), 46–49
value (Dynarray), 35	generator
value (Vector), 151	type (Random), 110
value (Word8Array), 164	get
value (Word8Vector), 166	value (Weak), 155
fromMicroseconds	getbool
value (Time), 147	value (Mysql), 84, 87
fromMilliseconds	value (Postgres), 105, 108
value (Time), 147	getc
fromPng	value (Substring), 139, 140
value (Gdimage), 43, 44	getCookie
fromReal	value (Mosmlcookie), 77
value (Time), 147	getCookieValue
fromSeconds	value (Mosmlcookie), 77
value (Time), 147	getcptr
fromString	value (Callback), 21, 22
value (Bool), 18	getCurrentLocation
value (Char), 24, 25	value (Location), 65 getdate
value (Date), 32, 33	value (Mysql), 84, 86
value (Int), 53, 54	value (Mysq1), 64, 60 value (Postgres), 105, 107
value (Path), 98, 100	getdatetime
value (Real), 114, 115	value (Mysql), 84, 86
value (String), 134, 135	value (Mysq1), 64, 66 value (Postgres), 105, 107
value (Time), 147, 148	getDir
value (Word), 158, 159	value (FileSys), 38
value (Word8), 161, 162	getdynfield
fromtag	value (Mysql), 85, 87
value (Mysql), 85	value (Postgres), 106, 108
value (Postgres), 106	getdyntup
fromTimeLocal	value (Mysql), 85, 87
value (Date), 32, 34	value (Postgres), 106, 108
fromTimeUniv	getdyntups
value (Date), 32, 34	value (Mysql), 85, 87
fromWord	value (Postgres), 106, 108
value (Signal), 124	qetEnv
ftype	value (Process), 109
value (Mysql), 85	getinetaddr
value (Postgres), 106	value (Socket), 126, 128
ftypes	getint
value (Mysql), 85	value (Mysql), 84, 86
value (Postgres), 106	value (Postgres), 105, 107
full	getItem
value (ArraySlice), 8	value (ArraySlice), 8, 9
value (CharArraySlice), 28	value (CharArraySlice), 28
value (CharVectorSlice), 30	value (CharVectorSlice), 30
value (Substring), 139, 140	value (List), 60, 61
value (VectorSlice), 153	value (VectorSlice), 153, 154
((. 333322230), 133, 131

value (Word8ArraySlice), 165	exception (NJ93), 88
value (Word8VectorSlice), 167	hd
getLexeme	value (List), 60
value (Lexing), 58, 59	value (NJ93), 88
getLexemeChar	head
value (Lexing), 58, 59	value (Msp), 78, 81
getLexemeEnd	Help (structure), 52
value (Lexing), 58, 59	help
getLexemeStart	value (Help), 52
value (Lexing), 58, 59	helpdirs
getOpt	value (Help), 52
value (Option), 92	host
getOrder	value (Mysql), 84, 85
value (Rbset), 111	value (Postgres), 105, 106
getParent	hour
value (Path), 98, 99	value (Date), 32, 33
getreal	hr
value (Mysql), 84, 86	value (Msp), 78, 81
value (Postgres), 105, 107	hra
getstring	value (Msp), 78, 81
value (Mysq1), 84, 86	html
value (Postgres), 105, 107	value (Msp), 78, 81 htmlcolors
gettime	
value (Mysql), 84, 86	value (Gdimage), 43, 44
value (Postgres), 105, 107	htmldoc
getTransparent	value (Msp), 78, 81
value (Gdimage), 43, 44	htmlencode
getVolume	value (Msp), 80, 83
value (Path), 98, 100	hup
Graphic	value (Signal), 124
exception (General), 46	
	Icase
h1	constructor (Regex), 120
value (Msp), 78, 81	ignore
h2	value (General), 46, 48
value (Msp), 78	ill
h3	value (Signal), 124
value (Msp), 78	image
h4	type (Gdimage), 43, 44
value (Msp), 78	value (Gdimage), 43, 44
h5	img
value (Msp), 78	value (Msp), 79, 82
h6	imga
value (Msp), 78	value (Msp), 79, 82
hash	implode
value (FileSys), 38, 40	value (NJ93), 88
value (Hashset), 50, 51	value (SML90), 123
value (Polyhash), 103, 104	value (String), 134, 135
value (Rbset), 111	
	in_flags type (Socket), 126, 129
hash_param	**
value (Polyhash), 103, 104	in_stream_length
hash_table	value (Nonstdio), 90
type (Polyhash), 103	inc
Hashset (structure), 50–51	value (NJ93), 88
hasKey	incheckbox
value (Gdbm), 41, 42	value (Msp), 79, 83
value (Polygdbm), 101	INCONSISTENT
Hd	constructor (PP), 93

indo	value (Mata) 60
indexfiles	value (Meta), 69
value (Help), 52	instream
inetAddr	type (BinIO), 12
value (Socket), 126, 127	type (NJ93), 88
inetDgram	type (SML90), 123
value (Socket), 126, 127	type (TextIO), 144
inetStream	insubmit
value (Socket), 126, 127	value (Msp), 79, 83
inhidden	Int (structure), 53–54
value (Msp), 79, 83	int
inpassword	type (General), 46, 47
value (Msp), 79, 83	type (Int), 53
input	value (Signal), 124
value (BinIO), 12	Interrupt
value (Msp), 79, 83	
·	exception (General), 46
value (NJ93), 88	intersection
value (SML90), 123	value (Binaryset), 16
value (TextIO), 144, 145	value (Intset), 56
input1	value (Rbset), 111, 112
value (BinIO), 12	value (Redblackset), 117
value (TextIO), 144, 145	value (Splayset), 132
input_binary_int	intext
value (Nonstdio), 90	value (Msp), 79, 83
input_char	Intmap (structure), 55
value (Nonstdio), 90	intmap
input_line	type (Intmap), 55
value (NJ93), 89	Intset (structure), 56-57
input_value	intset (shucture), 30-37
value (Nonstdio), 90	type (Intset), 56
inputa	intv
value (Msp), 79, 83	type (Rbset), 111
inputAll	Invalid_argument
value (BinIO), 12	exception (General), 46
value (TextIO), 144, 145	Io
inputc	exception (General), 46
value (NJ93), 89	isAbsolute
inputLine	value (Path), 98, 99
value (TextIO), 144, 145	isAlpha
inputN	value (Char), 24, 25
value (BinIO), 12, 13	isAlphaNum
value (TextIO), 144, 145	value (Char), 24, 25
inputNoBlock	isAscii
value (BinIO), 12	value (Char), 24, 25
value (TextIO), 144, 145	isCanonical
inradio	value (Path), 98, 99
value (Msp), 79, 83	isCntrl
inreset	value (Char), 24, 25
value (Msp), 79, 83	isdead
insert	value (Weak), 155, 156
value (Binarymap), 15	isDigit
value (Gdbm), 41, 42	value (Char), 24
value (Intmap), 55	isDir
value (Polygdbm), 101	value (FileSys), 38, 39
value (Polyhash), 103	isDst
value (Redblackmap), 116	value (Date), 32, 33
value (Splaymap), 131	isEmpty
installPP	value (ArraySlice), 8, 9
INDUCATIFF	value (Allayolice), 0, 9

value (Binaryset), 16	itemEnd
value (CharArraySlice), 28	value (Parsing), 96, 97
value (CharVectorSlice), 30	itemStart
value (Charvectors11ce), 30	value (Parsing), 96
	value (Farsing), 70
value (Intset), 56	join
value (Rbset), 111, 112	value (Option), 92
value (Redblackset), 117	joinBaseExt
value (Splayset), 132	value (Path), 98, 100
value (Substring), 139, 140	joinDirFile
value (VectorSlice), 153, 154	value (Path), 98, 100
value (Word8ArraySlice), 165	varae (1 acii), 70, 100
value (Word8VectorSlice), 167	kill
isGraph	value (Signal), 124
value (Char), 24, 25	value (Unix), 150
isHexDigit	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
value (Char), 24, 25	last
isLink	value (List), 60
value (FileSys), 38, 39	length
isLower	value (Array), 3
value (Char), 24	value (ArraySlice), 8
isnull	value (CharArray), 27
value (Mysql), 84, 87	value (CharArraySlice), 28
value (Postgres), 105, 108	value (CharVector), 29
isPrefix	value (CharVectorSlice), 30
value (String), 134, 135	value (List), 60
value (Substring), 139, 141	value (Vector), 151
isPrint	value (VectorSlice), 153
value (Char), 24, 25	value (Weak), 155, 156
isPunct	value (Word8Array), 164
value (Char), 24, 25	value (Word8ArraySlice), 165
isRegistered	value (Word8Vector), 166
value (Callback), 21, 22	value (Word8VectorSlice), 167
isRelative	lexbuf
value (Path), 98, 99	type (Lexing), 58
isSome	Lexing (structure), 58–59
value (Option), 92	li
isSpace	value (Msp), 79, 82
value (Char), 24, 25	liberal
isSubset	value (Meta), 69, 70
value (Binaryset), 16	List (structure), 60–61
value (Hashset), 50	list
value (Intset), 56	type (General), 46, 47
value (Rbset), 111, 112	type (List), 60
value (Redblackset), 117	listDir
value (Splayset), 132	value (Mosml), 73
isSubstring	listen
value (String), 134, 135	value (Socket), 126, 128
value (Substring), 139, 141	listItems
isSuccess	value (Binarymap), 15
value (Process), 109	value (Binaryset), 16, 17
isSuffix	value (Gdbm), 41, 42
value (String), 134, 135	value (Hashset), 50
value (Substring), 139, 141	value (Intmap), 55
isUpper	value (Intset), 56
value (Char), 24	value (Polygdbm), 101, 102
isweak	value (Polyhash), 103
value (Weak), 155	value (Rbset), 111, 112

1 (2 11 1) 116	1 (7) 70 92
value (Redblackmap), 116	value (Msp), 79, 82
value (Redblackset), 117, 118	mapEq
value (Splaymap), 131	value (ListPair), 62, 63
value (Splayset), 132, 133 listKeys	mapi
value (Gdbm), 41, 42	value (CharVector), 29
value (Polygdbm), 101, 102	value (CharVectorSlice), 30
ListPair (structure), 62–63	value (Vector), 151, 152
Listsort (structure), 64	value (VectorSlice), 153, 154 value (Word8Vector), 166
ln	
value (Math), 67	value (Word8VectorSlice), 167
value (NJ93), 88	mapMono
value (SML90), 123	value (Rbset), 111, 112
load	mapPartial
value (Meta), 69, 71	value (List), 60, 61
loaded	value (Option), 92
value (Meta), 69, 71	mark0
loadOne	value (Msp), 78, 81
value (Meta), 69, 71	mark0a
loadPath	value (Msp), 78, 81
value (Meta), 69, 71	markl
localOffset	value (Msp), 78, 81
value (Date), 32, 34	markla
Location	value (Msp), 78, 81
type (Location), 65	Match
Location (structure), 65–66	exception (General), 46
log10	Math (structure), 67–68
value (Math), 67	max
lookahead	value (Int), 53, 54
value (BinIO), 12, 13	value (NJ93), 88
value (NJ93), 88	value (Rbset), 111, 112
value (SML90), 123	value (Real), 114 value (Word), 158, 159
value (TextIO), 144, 145	
	value (Word8), 161, 162 maxChar
makestring	
value (General), 47, 49	value (Char), 24
map	maxInt
value (Binarymap), 15	value (Int), 53
value (CharVector), 29	maxLen
value (CharVectorSlice), 30	value (Chamberray) 27
value (Gdbm), 41, 42	value (CharArray), 27
value (Intmap), 55	value (CharVector), 29 value (Vector), 151
value (List), 60, 61	value (Vector), 131 value (Weak), 155, 156
value (ListPair), 62	
value (Msp), 79, 82	value (Word8Array), 164 value (Word8Vector), 166
value (Option), 92	maxOrd
value (Polygdbm), 101, 102	value (Char), 24
value (Polyhash), 103	maxSize
value (Rbset), 111, 112	value (String), 134
value (Redblackmap), 116	md5sum
value (Regex), 119, 121	value (Mosml), 73
value (Splaymap), 131	member
value (String), 134, 135 value (Vector), 151, 152	value (Binaryset), 16
value (Vector), 151, 152 value (VectorSlice), 153, 154	value (Binaryset), 10
value (VectorSIICe), 133, 134 value (Word8Vector), 166	value (Intset), 56
value (Word8Vector), 100 value (Word8VectorSlice), 167	value (Theset), 36 value (Rbset), 111, 112
	value (Roset), 111, 112 value (Redblackset), 117
mapa	varae (neubraensee), 117

value (Splayset), 132	value (Array2), 5, 7
merge	value (ArraySlice), 8, 10
value (Listsort), 64	value (CharArray), 27
mergeUniq	value (CharArraySlice), 28
value (Listsort), 64	value (Weak), 155, 157
Meta (structure), 69–72	value (Word8Array), 164
min	value (Word8ArraySlice), 165
value (Int), 53, 54	modTime
value (NJ93), 88	value (FileSys), 38, 39
value (Rbset), 111, 112	month
value (Real), 114	type (Date), 32
value (Word), 158, 159	value (Date), 32, 33
value (Word8), 161, 162	Mosml (structure), 73
minChar	Mosmlcgi (structure), 74-76
value (Char), 24	Mosmlcookie (structure), 77
minInt	Msp (structure), 78–83
value (Int), 53	Mysql (structure), 84-87
minute	
value (Date), 32, 33	name
mk ppstream	value (CommandLine), 31
value (PP), 93	nCols
mkAbsolute	value (Array2), 5, 6
value (Path), 98, 99	new
mkCanonical	value (Buffer), 19
value (Path), 98, 99	newgen
mkDict	value (Random), 110
value (Binarymap), 15	newgenseed
value (Belliar ymap), 13 value (Redblackmap), 116	value (Random), 110
	Newline
value (Splaymap), 131	constructor (Regex), 120
mkDir	nfields
value (FileSys), 38	value (Mysql), 84, 86
mkLoc	value (Postgres), 105, 107
value (Location), 65	nilLocation
mkPolyTable	value (Location), 65
value (Polyhash), 103, 104	NJ93 (structure), 88–89
mkRelative	Nl
value (Path), 98, 99	constructor (Msp), 80
mkTable	NO_RECVS
value (Polyhash), 103	constructor (Socket), 128
mod 1 (2 1) 47 40	NO_RECVS_OR_SENDS
value (General), 47, 49	constructor (Socket), 128
value (Int), 53	NO_SENDS
value (Word), 158, 159	constructor (Socket), 128
value (Word8), 161, 162	NonMonotonic
mode	exception (Rbset), 111
type (Gdimage), 43, 44	Nonstdio (structure), 90
modify	not
value (Array), 3, 4	value (Bool), 18
value (Array2), 5, 6	value (General), 46, 49
value (ArraySlice), 8,9	notb
value (CharArray), 27	value (Word), 158
value (CharArraySlice), 28	value (Word8), 161
value (Weak), 155, 156	Notbol
value (Word8Array), 164	constructor (Regex), 120
value (Word8ArraySlice), 165	notContains
modifyi	value (Char), 24, 25
value (Array), 3, 4	Noteol

constructor (Regex), 121	constructor (Postgres), 106
NotFound	type (Mysql), 84
exception (Binarymap), 15	type (Postgres), 105, 106
exception (Binaryset), 16	ol
exception (Gdbm), 41	value (Msp), 79, 82
exception (Hashset), 50	ola
exception (Intmap), 55	value (Msp), 79, 82
exception (Intset), 56	open_append
exception (Polygdbm), 101	value (NJ93), 89
exception (Rbset), 111	open_in
exception (Redblackmap), 116	value (NJ93), 88
exception (Redblackset), 117	value (SML90), 123
exception (Splaymap), 131	open_in_bin
exception (Splayset), 132	value (NJ93), 89
NotInt	value (Nonstdio), 90
exception (Msp), 78	open_out
noTransparent	value (NJ93), 88
value (Gdimage), 43, 45	value (SML90), 123
NotWriter	open_out_bin
exception (Gdbm), 41	value (NJ93), 89
exception (Polygdbm), 101	value (Nonstdio), 90
now	open_out_exe
value (Time), 147	value (Nonstdio), 90
nRows	openAppend
value (Array2), 5, 6	value (BinIO), 12, 13
nth	value (TextIO), 144, 145
value (List), 60	openbase
value (NJ93), 88	value (Mysql), 84, 85
nthtail	value (Postgres), 105, 106
value (NJ93), 88	openDir
ntuples	value (FileSys), 38
value (Mysql), 84, 86	openIn
value (Postgres), 105, 107	value (BinIO), 12
Null	value (TextIO), 12
exception (Mysql), 84	openmode
exception (Postgres), 105	type (Gdbm), 41
null	openOut
value (List), 60	value (BinIO), 12, 13
numItems	value (BIIIIO), 12, 13 value (TextIO), 144, 145
value (Binarymap), 15	Option Value (Text10), 144, 143
value (Binaryset), 16	exception (General), 46
value (Gdbm), 41, 42	
value (Hashset), 50	exception (Option), 92
value (Intmap), 55	Option (structure), 92
value (Intset), 56	option
value (Polygdbm), 101, 102	type (General), 46, 47
value (Polyhash), 103	type (Option), 92
value (Rbset), 111, 112	value (Msp), 79, 83
value (Redblackmap), 116	options
value (Redblackset), 117	value (Mysql), 84, 85
value (Splaymap), 131	value (Postgres), 105, 106
value (Splayset), 132	orb
	value (Word), 158
0	value (Word8), 161
value (General), 46, 48	ord
offset	value (Char), 24
value (Date), 32, 33	value (NJ93), 88
oid	value (SML90), 123

order	type (Parsing), 96
type (General), 46, 47	Parsing (structure), 96–97
ordof	part
value (NJ93), 88	type (Mosmlcgi), 74
orthodox	part_data
value (Meta), 69, 70	value (Mosmlcgi), 74, 75
OS (structure), 91	part_field_integer
OSAerr	
type (AppleScript), 2	value (Mosmlcgi), 74, 75
OSAID	part_field_string
	value (Mosmlcgi), 74, 75
type (AppleScript), 2	part_field_strings
out_flags	value (Mosmlcgi), 74,75
type (Socket), 126, 128	part_fieldnames
Out_of_memory	value (Mosmlcgi), 74,75
exception (General), 46	part_type
output	value (Mosmlcgi), 74
value (BinIO), 12, 13	partition
value (NJ93), 88	value (List), 60, 61
value (SML90), 123	passive
value (TextIO), 144, 145	
output1	type (Socket), 126, 127
value (BinIO), 12, 13	Path
value (TextIO), 124, 145	exception (Path), 98
	Path (structure), 98–100
output_binary_int	peek
value (Nonstdio), 90	value (Binarymap), 15
output_byte	value (Binaryset), 16
value (Nonstdio), 90	value (Gdbm), 41, 42
output_char	value (Hashset), 50
value (Nonstdio), 90	value (Intmap), 55
output_value	value (Polygdbm), 101
value (Nonstdio), 90	value (Polyhash), 103
outputc	
value (NJ93), 89	value (Redblackmap), 116
outputSubstr	value (Redblackset), 117
value (TextIO), 144	value (Splaymap), 131
outstream	value (Splayset), 132
type (BinIO), 12	peekInsert
type (Text10), 124	value (Polyhash), 103
Overflow	peekVal
exception (General), 46	value (Parsing), 96
exception (General), 40	pf_file
n	type (Socket), 126, 127
p walua (Man) 79, 91	pf_inet
value (Msp), 78, 81	type (Socket), 126, 127
pa 1 (22) 70 01	pi
value (Msp), 78, 81	-
packString	value (Math), 67
value (Byte), 20	pipe
padLeft	value (Signal), 124
value (StringCvt), 137	Polygdbm (structure), 101-102
padRight	Polyhash (structure), 103–104
value (StringCvt), 137, 138	polyHash
ParamMissing	value (Hashset), 50, 51
exception (Msp), 78	port
parentArc	value (Mysql), 84, 85
value (Path), 98, 99	value (Postgres), 105, 106
ParseError	pos_in
exception (Parsing), 96	value (Nonstdio), 90
• • • • • • • • • • • • • • • • • • • •	
parseTables	pos_out

1 (7 11) 00	1 (5 1) 110
value (Nonstdio), 90	value (Random), 110
position 120 141	Rbset (structure), 111–113
value (Substring), 139, 141	readDir
Postgres (structure), 105–108	value (FileSys), 38
pow (Y-1) 67	reader
value (Math), 67	type (StringCvt), 137
PP (structure), 93–95	readLink
pp_to_string	value (FileSys), 38, 39
value (PP), 93, 94	Real (structure), 114–115
ppconsumer	real
type (PP), 93	type (General), 46, 47
ppstream	type (Math), 67
type (General), 46, 47	type (Real), 114
pre	value (General), 47, 49
value (Msp), 79, 81	real_timer
precision	type (Timer), 149
value (Int), 53	realfmt
pred	type (StringCvt), 137
value (Char), 24	realPath
print	value (FileSys), 38, 39
value (NJ93), 88	reap
value (TextIO), 144, 146	value (Unix), 150
printDepth	recvArr
value (Meta), 69	value (Socket), 127, 129
printLength	recvArr'
value (Meta), 69	value (Socket), 127, 129
printseq	recvArrFrom
value (Msp), 78, 80	value (Socket), 127, 129
printVal	recvArrFrom'
value (Meta), 69	value (Socket), 127, 129
prmap	recvVec
value (Msp), 78, 80	value (Socket), 127, 128
proc	recvVec'
type (Unix), 150	
Process (structure), 109	value (Socket), 127, 129
prsep	recvVecFrom
value (Msp), 78, 80	value (Socket), 127, 129
	recvVecFrom'
quietdec	value (Socket), 127, 129
value (Meta), 69, 71	Redblackmap (structure), 116
quit	Redblackset (structure), 117–118
value (Meta), 69	ref
value (Signal), 124	constructor (General), 47
quot	type (General), 46, 47
value (Int), 53	regcomp
quotation	value (Regex), 119, 120
value (Meta), 69, 71	Regex
	exception (Regex), 119
radix	Regex (structure), 119–122
type (StringCvt), 137	regex
Random (structure), 110	type (Regex), 119
random	regexec
value (Random), 110	value (Regex), 119, 120
randomlist	regexecBool
value (Random), 110	value (Regex), 119, 121
range	region
value (Random), 110	type (Array2), 5, 6
rangelist	register
	-0910001

value (Callback), 21, 22	revAppend
regmatch	value (List), 60, 61
value (Regex), 119, 121	revfold
regmatchBool	value (NJ93), 88
value (Regex), 119, 121	rewindDir
regnexec	value (FileSys), 38
value (Regex), 119, 121	rgb
regnexecBool	type (Gdimage), 43, 44
value (Regex), 119, 121	value (Gdimage), 43, 44
rem	rmDir
value (Int), 53, 54	value (FileSys), 38
remove	round
value (Binarymap), 15	value (General), 47, 49
value (FileSys), 38, 39	value (Real), 114, 115
value (Gdbm), 41, 42	row
value (Intmap), 55	value (Array2), 5, 6
value (Polygdbm), 101	RowMajor
value (Polyhash), 103	constructor (Array2), 5
value (Redblackmap), 116	RTLD_LAZY
value (Splaymap), 131	constructor (Dynlib), 37
rename	RTLD_NOW
value (FileSys), 38, 39	constructor (Dynlib), 37
reorganize	run
value (Gdbm), 41, 42	value (Mosml), 73
value (GdSm), 41, 42 value (Polygdbm), 101, 102	runresult
replace	type (Mosml), 73
value (Regex), 119, 121	
replace1	sameDesc
value (Regex), 119, 121	value (Socket), 126, 129
replacer	sameSign
type (Regex), 119	value (Int), 53, 54
	value (Real), 114
reset value (Buffer), 19	scan
value (Mysq1), 84, 86	value (Bool), 18
value (Mysq1), 64, 60 value (Postgres), 105, 107	value (Date), 32, 34
resultstatus	value (Int), 53, 54
	value (Real), 114, 115
value (Mysq1), 84, 86 value (Postgres), 105, 107	value (Time), 147, 148
retrieve	value (Word), 158, 159
value (Binaryset), 16	value (Word8), 161, 162
value (Hashset), 50	scanStream
value (nashset), 50 value (Intmap), 55	value (TextIO), 144, 145
value (Redblackset), 117	scanString
	value (StringCvt), 137
value (Splayset), 132	second
rev	value (Date), 32, 33
value (List), 60	seek_in
revapp	value (Nonstdio), 90
value (Binarymap), 15	seek_out
value (Binaryset), 16, 17	value (Nonstdio), 90
value (Intmap), 55	segv
value (Intset), 56	value (Signal), 124
value (NJ93), 88	select
value (Rbset), 111, 112	value (Msp), 79, 83
value (Redblackmap), 116	value (Socket), 126, 129
value (Redblackset), 117, 118	sendArr
value (Splaymap), 131	value (Socket), 126, 128
value (Splayset), 132, 133	sendArr'

1 (2 1 1) 126 120	
value (Socket), 126, 128	size
sendArrTo	value (Buffer), 19
value (Socket), 126, 128	value (Gdimage), 43, 44
sendArrTo'	value (String), 134
value (Socket), 127, 128	value (Substring), 139, 140
sendVec	skipWS
value (Socket), 126, 128	value (StringCvt), 137
sendVec'	sleep
value (Socket), 126, 128	value (Process), 109
sendVecTo	slice
value (Socket), 126, 128	type (ArraySlice), 8
sendVecTo'	type (CharArraySlice), 28
value (Socket), 127, 128	type (CharVectorSlice), 30
set 100 100	type (VectorSlice), 153
type (Binaryset), 16	type (Word8ArraySlice), 165
type (Hashset), 50	type (Word8VectorSlice), 167
type (Rbset), 111	value (ArraySlice), 8
type (Redblackset), 117	value (CharArraySlice), 28
type (Splayset), 132	value (CharVectorSlice), 30
value (Weak), 155	value (Substring), 139, 140
setCookie	value (VectorSlice), 153
value (Mosmlcookie), 77	value (Word8ArraySlice), 165
setCookies	value (Word8VectorSlice), 167
value (Mosmlcookie), 77	SML90 (structure), 123
setTime	sock
value (FileSys), 38, 39	type (Socket), 126, 127
setTransparent	sock_addr
value (Gdimage), 43, 44	type (Socket), 126, 127
showquery	sock_desc
value (Mysql), 85, 87	type (Socket), 126
value (Postgres), 106, 108	sockDesc
shutdown	value (Socket), 126, 129
value (Socket), 126, 128	Socket (structure), 126–130
shutdown_mode type (Socket), 126	sort value (Arraysort), 11
**	value (Afraysort), 11 value (Listsort), 64
sign value (Int), 53, 54	sorted
value (Real), 114	value (Arraysort), 11
Signal (structure), 124–125	value (Listsort), 64
signal	Span
type (Signal), 124	exception (Substring), 139
type (Unix), 150	span
sin	value (Substring), 139, 141
value (Math), 67	specialfiles
value (NJ93), 88	value (Help), 52
value (M093), 88 value (SML90), 123	Splaymap (structure), 131
singleton	Splayset (structure), 132–133
value (Binaryset), 16	splitAt
value (Hashset), 50	value (Substring), 139, 141
value (Intset), 56	splitBaseExt
value (Rbset), 111	value (Path), 98, 100
value (Redblackset), 117	splitDirFile
value (Recurrackset), 117 value (Splayset), 132	value (Path), 98, 100
sinh	split1
value (Math), 67	value (StringCvt), 137
Size	value (Substring), 139, 141
exception (General), 46	splitr
exception (deneral), to	phirri

value (Substring), 139, 141	value (ArraySlice), 8
sqrt	value (CharArray), 27
value (Math), 67	value (CharArraySlice), 28
value (NJ93), 88	value (CharVector), 29
value (SML90), 123	value (CharVectorSlice), 30
startCPUTimer	value (Dynarray), 35
value (Timer), 149	value (Msp), 79, 82
startRealTimer	value (String), 134
value (Timer), 149	value (Substring), 139, 140
status	value (Vector), 151
type (Process), 109	value (VectorSlice), 153
value (Mysql), 84, 86	value (Weak), 155, 156
value (Postgres), 105, 107	value (Word8Array), 164
std err	value (Word8ArraySlice), 165
value (NJ93), 89	value (Word8Vector), 166
std in	value (Word8VectorSlice), 167
value (NJ93), 88	subArray
value (SML90), 123	value (Dynarray), 35
std_out	sublist
value (NJ93), 88	value (Rbset), 111, 113
value (SML90), 123	Subscript
stdErr	exception (General), 46
value (TextIO), 144, 146	subset
stdIn	value (Rbset), 111, 112
value (TextIO), 144, 145	subslice
stdOut	value (ArraySlice), 8, 9
value (TextIO), 144, 146	value (CharArraySlice), 28
stdoutPng	value (CharVectorSlice), 30
value (Gdimage), 43, 44	value (VectorSlice), 153
stop	value (Word8ArraySlice), 165
value (Signal), 124	value (Word8VectorSlice), 167
Str	substitute
constructor (Regex), 121	value (Regex), 119, 121
str	substitute1
value (String), 134, 135	value (Regex), 119, 121
stream	Substring
type (Socket), 126, 127	exception (NJ93), 88
streamsOf	Substring (structure), 139–142
value (Unix), 150	substring (structure), 137-142
String (structure), 134–136	type (General), 46, 47
string (structure), 134–130	type (Substring), 139
type (General), 46, 47	value (NJ93), 88
type (String), 134	value (String), 134
value (Gdimage), 43, 45	value (String), 134 value (Substring), 139
	,
value (Substring), 139, 140	Succ
StringCvt (structure), 137–138	value (Char), 24
stringToBytes	success
value (Byte), 20	value (Process), 109
stringUp	sup
value (Gdimage), 43, 45	value (Msp), 79, 82
strong	Sus
value (Msp), 79, 82	constructor (Regex), 121
style	Susp (structure), 143
type (Gdimage), 43, 44	susp
sub	type (Susp), 143
value (Array), 3	symbolEnd
value (Array2), 5, 6	value (Parsing), 96

1 10.	
symbolStart	exception (Time), 147
value (Parsing), 96	Time (structure), 147–148
symHandle	time
type (Dynlib), 36	type (Time), 147
SysErr	value (Mosml), 73
exception (General), 46	Timer (structure), 149
exception (OS), 91	title
syserror	value (Msp), 78, 81
type (General), 46, 47	tl
type (OS), 91	value (List), 60
system	value (NJ93), 88
value (Process), 109	tmpName
table	value (FileSys), 38, 39
type (Gdbm), 41	toCString
type (Gabin), 41 type (Polygdbm), 101	value (Char), 24, 26
value (Msp), 79, 82	value (String), 134, 135
tablea	toDefault
value (Msp), 79, 82	value (Real), 114
tabulate	toInt
value (Array), 3	value (Int), 53
value (Array2), 5, 6	value (Word), 158, 159
value (CharArray), 27	value (Word8), 161, 162
value (CharVector), 29	toIntX
value (Dynarray), 35	value (Word), 158, 159
value (List), 60, 61	value (Word8), 161, 162
value (Vector), 151	tokens
value (Word8Array), 164	value (Regex), 119, 122
value (Word8Vector), 166	value (String), 134, 135
take	value (Substring), 139, 142
value (List), 60	toLarge
takel	value (Int), 53
value (StringCvt), 137	toLargeInt
value (Substring), 139, 141	value (Word), 158, 160
taker	value (Word8), 161, 163
value (Substring), 139, 141	toLargeIntX
tan	value (Word), 158, 160
value (Math), 67	value (Word8), 161, 163
tanh	toLargeWord
value (Math), 67, 68	value (Word), 158, 160
td	value (Word8), 161, 163
value (Msp), 79, 82	toLargeWordX
tda	value (Word), 158, 160
value (Msp), 79, 82	value (Word8), 161, 163
term	toLower
value (Signal), 124	value (Char), 24, 25
terminate	toMicroseconds
value (Process), 109	value (Time), 147
textarea	toMilliseconds
value (Msp), 79, 83	value (Time), 147
textareaa	toPng
value (Msp), 79, 83	value (Gdimage), 43, 44
TextIO (structure), 144–146	toReal
th	value (Time), 147
value (Msp), 79, 82	toSeconds
tha	value (Time), 147
value (Msp), 79, 82	toString
Time	value (Bool), 18
	· ·

value (Char), 24, 25	value (Postgres), 105, 107
value (Date), 32, 33	
value (Int), 53, 54	ul
value (Path), 98, 100	value (Msp), 79, 82
value (Real), 114, 115	ula
value (String), 134, 135	value (Msp), 79, 82
value (Time), 147, 148	UnequalLengths
value (Word), 158, 159	exception (ListPair), 62
value (Word8), 161, 162	union
totalCPUTimer	value (Binaryset), 16
	value (Intset), 56
value (Timer), 149	value (Rbset), 111, 112
totalRealTimer	value (Redblackset), 117
value (Timer), 149	
toTime	value (Splayset), 132
value (Date), 32, 34	unit
toUpper	type (General), 46, 47
value (Char), 24, 25	Unix (structure), 150
toWord	unpackString
value (Signal), 124	value (Byte), 20
Tr	${\tt unpackStringVec}$
constructor (Regex), 121	value (Byte), 20
	unregister
tr . (v.) 70.82	value (Callback), 21, 22
value (Msp), 79, 82	unzip
tra	value (ListPair), 62
value (Msp), 79, 82	update
transform	value (Array), 3
value (Binarymap), 15	
value (Intmap), 55	value (Array2), 5, 6
value (Polyhash), 103, 104	value (ArraySlice), 8
value (Redblackmap), 116	value (CharArray), 27
value (Splaymap), 131	value (CharArraySlice), 28
translate	value (CharVector), 29
	value (Dynarray), 35
value (String), 134, 135	value (Vector), 151
value (Substring), 139, 141	value (Weak), 155, 156
traversal	value (Word8Array), 164
type (Array2), 5	value (Word8ArraySlice), 165
triml	value (Word8Vector), 166
value (Substring), 139, 140	urlencode
trimr	value (Msp), 80, 83
value (Substring), 139, 140	use (MSP), 60, 63
Trs	value (Meta), 69
constructor (Regex), 121	
trunc	usrl
value (General), 47, 49	value (Signal), 124
value (Real), 114, 115	usr2
	value (Signal), 124
truncate	
value (NJ93), 88	validVolume
tstp	value (Path), 98, 99
value (Signal), 124	valOf
tt	value (Option), 92
value (Msp), 79, 82	valuepoly
ttin	value (Meta), 69, 71
value (Signal), 124, 125	var
ttou	value (Callback), 21, 22
value (Signal), 124, 125	value (Dynlib), 36, 37
tty	vec2list
value (Mysql), 84, 85	value (Msp), 78, 80
ratae (11754±7, 0±, 05	varae (115p), 70, 00

vecDouble	word8
value (Mosml), 73	type (General), 46, 47
vecFloat	Word8Array (structure), 164
value (Mosml), 73	Word8ArraySlice (structure), 165
Vector (structure), 151–152	Word8Vector (structure), 166
vector (structure), 151 152	Word8VectorSlice (structure), 167
type (BinIO), 12	wordsize
type (CharArray), 27	value (Word), 158
type (CharArray), 27 type (CharArraySlice), 28	value (word), 136 value (Word8), 161
	· · · · · · · · · · · · · · · · · · ·
type (CharVector), 29 type (CharVectorSlice), 30	wseq type (Msp), 78, 80
· · · · · · · · · · · · · · · · · · ·	type (MSp), 78, 80
type (General), 46, 47	хL
type (Text10), 144	value (Location), 65
type (Vector), 151	xLR
type (Word8Array), 164	value (Location), 65
type (Word8ArraySlice), 165	xorb
type (Word8Vector), 166	value (Word), 158
type (Word8VectorSlice), 167	value (Word8), 161
value (Array), 3	xR
value (ArraySlice), 8, 9	
value (CharArray), 27	value (Location), 65
value (CharArraySlice), 28	XXLR
value (CharVectorSlice), 30	value (Location), 65
value (General), 47, 49	XXRL
value (VectorSlice), 153, 154	value (Location), 65, 66
value (Word8Array), 164	ху
value (Word8ArraySlice), 165	type (Gdimage), 43, 44
value (Word8VectorSlice), 167	TOOK
vector_slice	year value (Date), 32, 33
type (CharArraySlice), 28	
type (Word8ArraySlice), 165	yearDay
VectorSlice (structure), 153-154	value (Date), 32, 33
verbose	yyexit
value (Meta), 69, 71	exception (Parsing), 96
	yyparse
Weak (structure), 155–157	value (Parsing), 96
weak	zeroTime
type (Weak), 155	value (Time), 147
value (Weak), 155	zip
weekDay	value (ListPair), 62
value (Date), 32, 33	zipEq
weekday	value (ListPair), 62, 63
type (Date), 32	value (histrail), 02, 03
welcome	
value (Help), 52	
with_pp	
value (PP), 93, 94	
withtable	
value (Gdbm), 41	
value (Polygdbm), 101	
withtables	
value (Gdbm), 41	
Word (structure), 158–160	
word	
type (General), 46, 47	
type (Word), 158	
type (Word8), 161	
Word8 (structure), 161–163	
,,	