Standard ML of New Jersey

—

Tools

(Version 0.93)

February 15, 1993

Copyright © 1993 by AT&T Bell Laboratories

Contents

NAME

Intro — introduction to SML software tools

DESCRIPTION

The directory `tools` contains several tools to assist in building and debugging SML programs. The current set of tools consists of

Concurrent ML
a threads library

debugger
a replay debugger

eXene
a multi-threaded X Window System toolkit

info
an ultra-simple environment browser

makeml
shell script for building Standard ML of New Jersey

mllex
a lexical analyzer generator

mltwig
a code-generator generator

mlyacc
a parser generator

profile
an execution time profiler

SourceGroup
a separate compilation and make system

Most of the man pages in this chapter are mere pointers to a tool and its documentation in the appropriate subdirectory of `tools`.

NAME

CML — Concurrent ML

DESCRIPTION

CML is a library providing threads and synchronous communication via typed chanels. See the documentation in `CML-0.9.8` for details.

SEE ALSO

"CML: A Higher-order Concurrent Language," *Proceedings of the SIGPLAN '91 Conference on Programming Language Design and Implementation.*

NAME

Debugger — replay debugger with time-travel

DESCRIPTION

The debugger supports conventional, and some unconventional debugging capabilities, based on instrumentation and "time travel". A special debugging version of the interactive system, **smld**, must be built and installed (see the **makeml** man page), and this is used in conjunction with a gnu emacs interface available in `tools/debug`. See the debugger documentation (in `tools/debug`) for details.

NAME

eXene — Multi-threaded X Windows library

DESCRIPTION

eXene is a multi-threaded X Window System toolkit, which is implemented in Concurrent ML. See the documentation in `eXene-0.4` for details.

SEE ALSO

CML(TOOL)

NAME

Info — a very simple browser for environments

SYNOPSIS

signature INFO
structure Info : Info

SIGNATURE

```
exception Info
val info   : string list -> unit
val search : string -> unit
```

DESCRIPTION

Info

raised if no information is found (*i.e.* the identifier is not bound in any form in the environment).

info

takes the full path of an object and search the top level environment for information on that object (of the same kind as is displayed when the object was defined).

search

takes a string and looks for all visible bindings containing that string as a subpart. It walks through the full environment tree of the top level environment

EXAMPLE

```
- Info.info ["+"];
val + : 'a * 'a -> 'a as + : int * int -> int + : real * real -> real
infix 6 +
val it = () : unit
- Info.info ["Array"];
structure Array :
  sig
    eqtype 'a array
    exception Size
    exception Subscript
    val array : int * '1a -> '1a array
    val arrayoflist : '1a list -> '1a array
    val length : 'a array -> int
    val sub : 'a array * int -> 'a
    val tabulate : int * (int -> '1a) -> '1a array
    val update : 'a array * int * 'a -> unit
  end
val it = () : unit
- Info.search "compile";
variable or constructor NewJersey.System.Compile.compile
variable or constructor NewJersey.System.Compile.compileAst
variable or constructor System.Compile.compile
variable or constructor System.Compile.compileAst
val it = () : unit
```

FILES

tools/info/info.sml

SEE ALSO

Env(SYS)

CAVEATS

This is based on Env(SYS), but it needs somewhat more sophisticated primitives to ensure complete accuracy. This is a simple first step toward a more capable environment browser.

NAME

makeml — build the Standard ML of New Jersey system

SYNOPSIS

makeml *options*

DESCRIPTION

Makeml is a tool for building the Standard ML of New Jersey system (SML/NJ) from source and ML-object ('.mo') files. SML/NJruns on a number of machine architectures (MC680x0, Mips, SPARC, RS/6000, HPPA, and i386/i486) and under a number of different operating systems (SunOS, 4.3bsd, Mach, IRIX, Ultrix, AIX, ...). There are also several different configurations of the system that can be built. Makeml provides a reasonable interface to these various options.

OPTIONS

The following options are used to specify the machine and operating system configuration. These are the only ones necessary for the basic installation.

-sun3 *(sunos | mach)*
: Build the system for the Sun 3.

-sun4 *(sunos | mach | solaris)*
: Build the system for sparc machines, including the Sparcstation 10.

(-rs6000 | -rs6k) *aix*
: Build the version for the IBM RS/6000 workstations. **Note**: this requires AIX version 3.2.

-decstation *(bsd | ultrix | mach)*
: Build the version for the DEC mips processor boxes. These are little-endian machines.

-mips *(riscos | mach)*
: Build the version for the MipsCo machines (R3000, R6280). This is a big-endian machine.

-sgi *(irix | irix3)*
: Build the version for the Silicon Graphics machines; the **irix3** option specifies Irix 3.x, otherwise Irix 4.x is assumed. These are big-endian mips processors.

-hppa *hpux8*
: Build the hppa version running under HPUX 8.0 (earlier versions of HPUX have not been tested). By default makeml builds a noshare version (see **-noshare** option), and the **-pervshare** option is ignored.

-m68 *(aux | sunos | mach | hpux | hpux8 | more)*
: Build a version for a M680x0 family machine. The **hpux8** option is for version 8.0 of the HPUX operating system; use **hpux** for earlier versions.

-next *(2 | 3)*
: Build the version for the NeXT machine (either NeXTstep 2.x or NeXTstep 3.x). The NeXT machine uses a non-standard version of MACH as its operating system, which isn't BSD compatible.

-i386 *(mach | bsd | bsd386)*
: Build the system for i386/i486 machines. The bsd386 version has patches to fix problems with signals in BSD/386.

-sequent *dynix3*
: Build the system for the Sequent (i386).

-vax *(bsd | mach)*
Build the vax version. This version is "out of service" for version 0.93 of SML/NJ. Use version 0.75 on the vax.

The following options are used to specify the kind of system to build.

-debug Build an image (with default name 'smld') with the debugger loaded.

-i Make the 'sml' image start out using the interpreter for faster compilation and slower execution (for interactive system only; can switch back to native code once in 'sml' by 'System.Control.interp := false').

-ionly Build an image (with default name 'smli') that has only the interpreter. This gives fast compilation and saves space by eliminating the code generator from the executable, but results in slower execution.

-batch Build the batch compiler (with default name 'smlc') instead of an interactive system.

-target *machine*
Build a batch cross compiler for *machine*. Valid machine names are: **m68**, **sparc**, **mipsl**, **mipsb**, **vax**, and **i386**. Note that for the Mips architecture you must specify the endianess. This option implies the **-batch** option.

-o *imag*
Use image as the name of the system image. The default image name is 'sml' for interactive systems, 'smld' for the debugger version, 'smli' for the interpreter only system and 'smlc' for the batch compiler.

-noshare
Do not link the '.mo' files into an 'a.out' format object file and include it in the runtime executable.

-pervshare
Link only a minimal set of '.mo' files into the object. This is not applicable to the HPPA.

-gcc Use the GNU C compiler to compile the run-time system. This will improve the garbage collector performance on some machines (e.g., Sun3). **Note:** this only works with GCC 1.xx.

The following options may be used to tune garbage collection and paging performance.

-h *heapsize*
Set the initial heap size to *heapsize* kilo-bytes.

-m *softlimit*
Set the soft limit on the heap size to *softlimit* kilo-bytes.

-r *ratio* Set the ratio of the heap size to live data to *ratio*, which must be an integer value no less than 3.

The following options are for building and testing new versions of the system; they are not necessary for normal installation.

-run Build the run-time kernel ('runtime/run'), but don't build a system.

-noclean
Don't remove the existing '.o' files in the runtime directory.

-norun
Don't re-compile the runtime kernel. This implies the **-noclean** option.

-mo *path*
Use *path* as the directory containing the '.mo' files.

-runtime *path*
Use *path*] as the source directory for the runtime code.

-g Compile the runtime with the **-g** command line option.

-D *def* When compiling the runtime code add "**-D** *def*" as a command line option.

-debug0
Build a version with the debugger internals, but not the user-level code.

USAGE

For the standard configuration, the only options required are the machine type and operating system. For example

```
makeml -sun4 sunos
```

builds the SPARC version of the interactive system to run on SunOS systems. Another example is

makeml -sun4 sunos -target mipsl

which builds a **sparc** to **mipsl** cross compiler.

ENVIRONMENT

GCC

Specifies the path of **gcc**. Set this if your path doesn't contain **gcc** and you are using the '**-gcc**' option.

FILES

src/makeml	the makeml shell script.

SEE ALSO

makeml(1), linkdata(1), sml(1)

NAME

ML-Lex — lexical analyzer generator for ML

DESCRIPTION

ML-Lex is a lexical analyzer generator implemented in ML. It accepts semantic actions in ML and generates an ML program. See the ML-Lex documentation (in `tools/lexgen`) for details.

NAME

ML-Twig — code-generator generator for ML

DESCRIPTION

ML-Twig is a code-generator generator implemented in ML. It accepts semantic actions in ML and generates an ML program. See the ML-Twig documentation (in `tools/mltwig`) for details.

NAME

ML-Yacc — parser generator for ML

DESCRIPTION

ML-Yacc is a robust, error-correcting parser generator implemented in ML. It accepts semantic actions in ML and generates an ML program. See the ML Yacc documentation (in `tools/mlyacc`) for details.

NAME

Profile — execution time profiler

SYNOPSIS

smlp

signature PROFILE
structure Profile : PROFILE

SIGNATURE

```
val profiling : bool ref
val profileOn : unit -> unit
val profileOff : unit -> unit
val reset : unit -> unit
val report : outstream -> unit
```

DESCRIPTION

Statistical profile of the execution time of ML programs: approximate time spent in each function, and the number of times each function is called.

First, construct an **smlp** executable (if your system administrator has not already done so) by executing the shell command:
`sml <tools/profile/profile_script`
Then load and execute your program in **smlp** as you would in **sml**. If you don't want compiling time counted in the profile report, execute `Profile.reset()` after compiling (but before executing) your program. Finally, generate a report with the command `Profile.report std_out`.

profiling
Controls whether newly compiled code should have profiling instrumentation. Default is **true**.

profileOn ()
Turn the interrupt timer on (off) to gather execution-time data. Default is on.

profileOff ()
Turn the interrupt timer off.

reset ()
Set all call counts and time accumulations back to zero.

report *outstream*
Generate a report on *outstream*.

The report has the following format:

```
 %time cumsec   #call  name
 43.64   5.73 1243148  try
 23.99   8.88 1277673  try.while
 21.62  11.72       0  Garbage Collection
  7.76  12.74       0  Other
  1.75  12.97 1243134  Ref.inc
   .60  13.05       1  listofarray.loa
   .60  13.13       0  Compilation
```

The list is sorted in decreasing order of time spent in each function. Some of the items listed (Garbage Collection, Compilation, Other) are system overhead, not ML functions. An actual report will be much longer, as it lists all library functions (called or not).

%time is the percentage of time spent in each function.

cumsec is the cumulative number of sections spent in this function and all the ones listed above it.

#call is the number of calls to this function.

name is the name of the function, prefixed by the names of functions and structures in which it is nested. Anonymous functions (built using **fn**, not **fun**) are listed as `anon` and can be disambiguated by prefix.

FILES

tools/profile/profile_script   commands to build **smlp**

CAVEATS

Slows down program execution by a factor of three or so .

Mixing profiled code (compiled with Profile.profiling = true) with unprofiled code (compiled with Profile.profiling = false) leads to inaccurate results.

NAME

SourceGroup — separate compilation system for ML

DESCRIPTION

SourceGroup is a separate compilation and "make" facility for Standard ML of New Jersey.

See the SourceGroup documentation (in `tools/sourcegroup`) for details.