

```
1: # $Id: README,v 1.3 2020-09-06 20:13:06-07 - - $
2:
3: NAME
4:   mbtran - translator from MB to MBIR
5:
6: SYNOPSIS
7:   mbtran mbsource >mbiobject
8:
9: DESCRIPTION
10:  The mbtran utility translate Mini Basic source code into
11:  Mini Basic Intermediate Representation, which looks like
12:  Scheme. You do not need to understand scanning, parsing,
13:  or O'Caml to use this. See the MBIR sources in another
14:  directory.
15:
```

```
1: (* $Id: etc.mli,v 1.1 2009-02-03 17:24:36-08 - - $ *)
2:
3: (*
4:  * Main program and system access.
5:  *)
6:
7: val execname : string
8:
9: val quit : unit -> unit
10:
11: val eprint : string list -> unit
12:
13: val lexepprint : Lexing.position -> string list -> unit
14:
15: val usageprint : string list -> unit
16:
```

```
1: (* $Id: absyn.ml,v 1.3 2020-09-06 20:13:06-07 - - $ *)
2:
3: (*
4:  * Abstract syntax definitions for MB->MBIR.
5:  *)
6:
7: type linenr      = int
8: and variable     = string
9: and label        = string
10: and number       = float
11: and oper         = string
12: and array        = variable * expr
13: and fncall       = variable * expr
14:
15: and print        = Printexpr of expr
16:                  | String of string
17:
18: and memref       = Array of array
19:                  | Variable of variable
20:
21: and expr         = Binop of oper * expr * expr
22:                  | Unop of oper * expr
23:                  | Memref of memref
24:                  | Constant of number
25:                  | Fncall of fncall
26:
27: and stmt         = Dim of array
28:                  | Let of memref * expr
29:                  | Goto of label
30:                  | If of expr * label
31:                  | Print of print list
32:                  | Input of memref list
33:
34: and program      = (linenr * label option * stmt option) list
35:
```

```
1: (* $Id: etc.ml,v 1.1 2009-02-03 17:24:36-08 - - $ *)
2:
3: open Lexing
4: open Printf
5:
6: let execname = Filename.basename Sys.argv.(0)
7:
8: let exit_code_ref = ref 0
9:
10: let quit () = exit !exit_code_ref
11:
12: let eprintlist message =
13:   (exit_code_ref := 1;
14:    flush_all ();
15:    List.iter (fprintf stderr "%s") message;
16:    fprintf stderr "\n";
17:    flush_all ())
18:
19: let eprint message = eprintlist (execname :: ": " :: message)
20:
21: let lexepprint position message =
22:   eprint (position.pos_fname :: ": "
23:           :: string_of_int position.pos_lnum :: ": "
24:           :: message)
25:
26: let usageprint message =
27:   eprintlist ("Usage: " :: execname :: " " :: message)
28:
```

```
1: (* $Id: main.ml,v 1.4 2020-09-06 20:06:16-07 - - $ *)
2:
3: open Absyn
4: open Etc
5: open Lexing
6: open Printf
7:
8: (*
9:  * Functions for printing out the absyn tree in Scheme format.
10: *)
11:
12: let
13:   rec pr'print file print = match print with
14:   | Printexpr (expr)          -> fprintf file " %a" pr'expr expr
15:   | String (string)          -> fprintf file " %s" string
16:
17:   and pr'memref file memref = match memref with
18:   | Array (array)            -> pr'array file array
19:   | Variable (variable)      -> fprintf file "%s" variable
20:
21:   and pr'expr file expr = match expr with
22:   | Binop (oper, expr1, expr2) -> fprintf file "(%s %a %a)"
23:                                   oper pr'expr expr1 pr'expr expr2
24:   | Unop (oper, expr)          -> fprintf file "(%s %a)"
25:                                   oper pr'expr expr
26:   | Memref (memref)            -> pr'memref file memref
27:   | Constant (number)          -> fprintf file "%.15g" number
28:   | Fncall (fncall)           -> pr'fncall file fncall
29:
30:   and pr'stmt file stmt = match stmt with
31:   | None                      -> ()
32:   | Some (Dim (array))        -> fprintf file "(dim %a)"
33:                                   pr'array array
34:   | Some (Let (memref, expr)) -> fprintf file "(let %a %a)"
35:                                   pr'memref memref pr'expr expr
36:   | Some (Goto (label))        -> fprintf file "(goto %s)" label
37:   | Some (If (expr, label))    -> fprintf file "(if %a %s)"
38:                                   pr'expr expr label
39:   | Some (Print (prints))      -> fprintf file "(print%a)"
40:                                   pr'prints prints
41:   | Some (Input (memrefs))     -> fprintf file "(input%a)"
42:                                   pr'inputs memrefs
43:
44:   and pr'array file (variable, expr) =
45:     fprintf file "(asub %s %a)" variable pr'expr expr
46:
47:   and pr'fncall file (ident, expr) =
48:     fprintf file "(%s %a)" ident pr'expr expr
49:
50:   and pr'line file (line, label, stmt) =
51:     let s'label = match label with
52:     | None          -> ""
53:     | Some (label)  -> label
54:     in fprintf file "(%5d %-8s %a)\n" line s'label pr'stmt stmt
55:
56:   and pr'input   file inputs  = fprintf file " %a" pr'memref inputs
57:
58:   and pr'prints  file prints  = List.iter (pr'print file) prints
```

```
59:
60: and pr'inputs  file memrefs = List.iter (pr'input file) memrefs
61:
62: and pr'lines   file lines   = List.iter (pr'line file) lines
63:
64: and pr'program file program = fprintf file "(\n%a)\n" pr'lines program
65:
66: (*
67:  * Main program reads a file and prints to stdout.
68:  *)
69:
70: let translatefile filename =
71:   try (printf ";;Source: %s\n" filename;
72:        let sourcefile =
73:          if filename = "-" then stdin else open_in filename in
74:          let lexbuf = Lexing.from_channel sourcefile in
75:          let absyn = Parser.program Lexer.token lexbuf in
76:          printf "%a" pr'program absyn)
77:   with Sys_error (string) -> eprint [string]
78:
79: let _ = if !Sys.interactive
80:   then ()
81:   else match Array.length Sys.argv with
82:   | 1 -> translatefile "-"
83:   | 2 -> translatefile Sys.argv.(1)
84:   | _ -> usageprint ["[filename.mb]"]
85:
```

```
1: /* $Id: parser.mly,v 1.2 2019-11-26 14:07:46-08 - - $ */
2:
3: %{
4: (***** BEGIN PARSER SEMANTICS *****)
5:
6: open Absyn
7: open Etc
8: open Lexing
9:
10: let syntax () = lexepri (symbol_start_pos ()) ["syntax error"]
11:
12: let linenr () = (symbol_start_pos ()).pos_lnum
13:
14: (***** END PARSER SEMANTICS *****)
15: %}
16:
17: %token <string> RELOP EQUAL ADDOP MULOP POWOP
18: %token <string> IDENT NUMBER STRING
19: %token COLON COMMA LPAR RPAR LSUB RSUB EOL EOF
20: %token DIM LET GOTO IF PRINT INPUT
21:
22: %type <Absyn.program> program
23:
24: %start program
25:
26: %%
27:
28: program : stmts EOF                { List.rev $1 }
29:
30: stmts   : stmts stmt EOL           { $2::$1 }
31:         | stmts error EOL          { syntax (); $1 }
32:         |                          { [] }
33:
34: stmt    : label action              { (linenr (), Some $1, Some $2) }
35:         | action                    { (linenr (), None, Some $1) }
36:         | label                      { (linenr (), Some $1, None) }
37:         |                          { (linenr (), None, None) }
38:
39: label   : ident COLON               { $1 }
40:
41: action  : DIM array                 { Dim ($2) }
42:         | LET memref EQUAL expr     { Let ($2, $4) }
43:         | GOTO ident                { Goto ($2) }
44:         | IF relexpr GOTO ident     { If ($2, $4) }
45:         | PRINT prints               { Print ($2) }
46:         | PRINT                     { Print ([]) }
47:         | INPUT inputs               { Input ($2) }
48:
49: prints  : print COMMA prints        { $1::$3 }
50:         | print                      { [$1] }
51:
52: print   : expr                      { Printexpr ($1) }
53:         | STRING                     { String ($1) }
54:
55: inputs  : memref COMMA inputs       { $1::$3 }
56:         | memref                     { [$1] }
57:
58: memref  : ident                     { Variable ($1) }
```

```
59:      | array                { Array ($1) }
60:
61: array  : ident LSUB expr RSUB { ($1, $3) }
62:
63: relexpr : expr RELOP expr    { Binop ($2, $1, $3) }
64:      | expr EQUAL expr      { Binop ($2, $1, $3) }
65:
66: expr    : expr ADDOP term     { Binop ($2, $1, $3) }
67:      | term                    { $1 }
68:
69: term    : term MULOP factor   { Binop ($2, $1, $3) }
70:      | factor                  { $1 }
71:
72: factor  : primary POWOP factor { Binop ($2, $1, $3) }
73:      | primary                 { $1 }
74:
75: primary : LPAR expr RPAR      { $2 }
76:      | ADDOP primary          { Unop ($1, $2) }
77:      | NUMBER                  { Constant (float_of_string ($1)) }
78:      | memref                  { Memref ($1) }
79:      | ident LPAR expr RPAR    { Fncall ($1, $3) }
80:
81: ident   : IDENT               { $1 }
82:      | DIM                     { "dim" }
83:      | GOTO                    { "goto" }
84:      | IF                      { "if" }
85:      | INPUT                   { "input" }
86:      | LET                     { "let" }
87:      | PRINT                   { "print" }
88:
```



```
1: (* $Id: lexer.mll,v 1.5 2020-08-30 19:36:37-07 - - $ *)
2:
3: {
4: (***** BEGIN LEXER SEMANTICS *****)
5:
6: open Absyn
7: open Etc
8: open Lexing
9: open Parser
10: open Printf
11:
12: let lexerror lexbuf =
13:     lexepri (lexeme_start_p lexbuf)
14:     ["invalid character '" ^ (lexeme lexbuf) ^ "'"]
15:
16: let newline lexbuf =
17:     let incrline pos =
18:         {pos with pos_lnum = pos.pos_lnum + 1; pos_bol = pos.pos_cnum}
19:     in (lexbuf.lex_start_p <- incrline lexbuf.lex_start_p;
20:         lexbuf.lex_curr_p <- incrline lexbuf.lex_curr_p)
21:
22: (*
23: let list lexbuf =
24:     let pos = lexeme_start_p lexbuf
25:     in (if pos.pos_bol = pos.pos_cnum
26:         then printf ";;%4d: " pos.pos_lnum;
27:         printf "%s" (lexeme lexbuf))
28: *)
29:
30: (***** END LEXER SEMANTICS *****)
31: }
32:
33: let letter      = ['a'-'z' 'A'-'Z' '_' ]
34: let digit       = ['0'-'9' ]
35: let fraction    = (digit+ '.'? digit* | '.' digit+)
36: let exponent    = (['E' 'e'] ['+' '-']? digit+)
37:
38: let comment     = ('#' ['^'\n']* )
39: let ident       = (letter (letter | digit)*)
40: let number      = (fraction exponent?)
41: let string      = ('"' ['^'\n' '\t']* '"')
42:
43: rule token      = parse
44:     eof          { EOF }
45:     [' ' '\t']   { token lexbuf }
46:     comment      { token lexbuf }
47:     "\n"         { newline lexbuf; EOL }
48:     ":"          { COLON }
49:     ","          { COMMA }
50:     "("          { LPAR }
51:     ")"          { RPAR }
52:     "["          { LSUB }
53:     "]"          { RSUB }
54:     "="          { EQUAL (lexeme lexbuf) }
55:     "!="         { RELOP (lexeme lexbuf) }
56:     "<"          { RELOP (lexeme lexbuf) }
57:     "<="         { RELOP (lexeme lexbuf) }
58:     ">"          { RELOP (lexeme lexbuf) }
```

```
59:      ">="      { RELOP (lexeme lexbuf) }
60:      "+"      { ADDOP (lexeme lexbuf) }
61:      "-"      { ADDOP (lexeme lexbuf) }
62:      "*"      { MULOP (lexeme lexbuf) }
63:      "/"      { MULOP (lexeme lexbuf) }
64:      "%"      { MULOP (lexeme lexbuf) }
65:      "^"      { POWOP (lexeme lexbuf) }
66:      "dim"     { DIM }
67:      "goto"    { GOTO }
68:      "if"      { IF }
69:      "input"   { INPUT }
70:      "let"     { LET }
71:      "print"   { PRINT }
72:      number    { NUMBER (lexeme lexbuf) }
73:      string    { STRING (lexeme lexbuf) }
74:      ident     { IDENT (lexeme lexbuf) }
75:      _         { lexerror lexbuf; token lexbuf }
76:
```

```
1: # $Id: Makefile,v 1.13 2020-09-06 20:13:06-07 - - $
2:
3: #
4: # General useful macros
5: #
6:
7: MKFILE      = Makefile
8: MAKEFLAGS += --no-builtin-rules
9: DEPSFILE    = ${MKFILE}.deps
10: NOINCLUDE   = ci clean spotless
11: NEEDINCL    = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
12:
13: #
14: # File list macros
15: #
16:
17: EXECBIN      = mbtran
18: OBJCMX       = absyn.cmx etc.cmx parser.cmx lexer.cmx main.cmx
19: OBJCMI       = ${patsubst %.cmx, %.cmi, ${OBJCMX}}
20: OBJBIN       = ${patsubst %.cmx, %.o, ${OBJCMX}}
21: MLSOURCE     = etc.mli absyn.ml etc.ml main.ml
22: GENSOURCE    = parser.mli parser.ml lexer.ml
23: GENFILES     = ${GENSOURCE} parser.output
24: ALLSOURCES   = README ${MLSOURCE} parser.mly lexer.mli ${MKFILE}
25:
26: #
27: # General targets
28: #
29:
30: all : ${EXECBIN}
31:
32: ${EXECBIN} : ${OBJCMX}
33:             ocamlpt ${OBJCMX} -o ${EXECBIN}
34:
35: %.cmi : %.mli
36:             ocamlc -c $<
37:
38: %.cmx : %.ml
39:             ocamlpt -c $<
40:
41: %.ml : %.mli
42:             ocamllex $<
43:
44: %.mli %.ml : %.mly
45:             ocamlyacc -v $<
46:
47: #
48: # Misc targets
49: #
50:
51: clean :
52:         - rm ${OBJCMI} ${OBJCMX} ${OBJBIN}
53:
54: spotless : clean
55:         - rm ${EXECBIN} ${GENFILES} Listing.ps Listing.pdf
56:
57: ci : ${ALLSOURCES}
58:         cid -is ${ALLSOURCES}
```

```
59:
60: deps : ${MLSOURCE} ${GENSOURCE}
61:         @ echo "# ${DEPSFILE} created `date`" >${DEPSFILE}
62:         ocamldep ${MLSOURCE} ${GENSOURCE} >>${DEPSFILE}
63:
64: ${DEPSFILE} :
65:         @touch ${DEPSFILE}
66:         ${MAKE} deps
67:
68: lis : ${ALLSOURCES}
69:         mkpspdf Listing.ps ${ALLSOURCES}
70:
71: again :
72:         ${MAKE} spotless
73:         ${MAKE} deps
74:         ${MAKE} ci
75:         ${MAKE} all
76:         ${MAKE} lis
77:
78: ifeq "${NEEDINCL}" ""
79: include ${DEPSFILE}
80: endif
81:
```