CHAPTER 13

MASTERSCOPE

Masterscope is an interactive program for analyzing and cross referencing user programs. It contains facilities for analyzing user functions to determine what other functions are called, how and where variables are bound, set, or referenced, as well as which functions use particular record declarations. Masterscope is able to analyze de nitions directly from a le as well as in-core de nitions.

Masterscope maintains a database of the results of the analyses it performs. Via a simple command language, the user may interrogate the database, call the editor on those expressions in functions that were analyzed which use variables or functions in a particular way, or display the tree structure of function calls among any set of functions.

Masterscope is interfaced with the editor and le package so that when a function is edited or a new de nition loaded in, Masterscope knows that it must re-analyze that function.

The following sample session illustrates some of these facilities.

```
_. ANALYZE FUNCTIONS ON RECORD                      [1]
..............................                      [2]
NIL
_. WHO CALLS RECFIELDLOOK                           [3]
(RECFIELDLOOK ACCESSDEF ACCESSDEF2 EDITREC)
_. EDIT WHERE ANY CALL RECFIELDLOOK                 [4]
RECFIELDLOOK :
(RECFIELDLOOK (CDR Y) FIELD)
tty:
*OK
ACCESSDEF :
(RECFIELDLOOK DECLST FIELD VAR1)
*OK
(RECFIELDLOOK USERRECLST FIELD)
*N VAR1
*OK
ACCESSDEF2 :
(RECFIELDLOOK (RECORD.SUBDECS TRAN) FIELD)
tty:
(RECFIELDLOOK (RECORD.SUBDECS TRAN) FIELD)
*N (CAR TAIL]
*OK
EDITREC :
(RECFIELDLOOK USERRECLST (CAR EDITRECX))
*OK
NIL
_. WHO CALLS ERROR                                  [5]
..
```

```
(EDITREC)
_. SHOW PATHS TO RECFIELDLOOK FROM ACCESSDEF        [6]
(inverted tree)


1. RECFIELDLOOK RECFIELDLOOK
2.           ACCESSDEF
3.           ACCESSDEF2 ACCESSDEF2
4.                     ACCESSDEF
5.                     RECORDCHAIN ACCESSDEF
NIL
_. WHO CALLS WHO IN /FNS                            [7]
RECORDSTATEMENT --  /RPLNODE
RECORDECL1 --       /NCONC, /RPLACD, /RPLNODE
RECREDECLARE1 --    /PUTHASH
UNCLISPTRAN --      /PUTHASH, /RPLNODE2
RECORDWORD --       /RPLACA
RECORD1 --          /RPLACA, /SETTOPVAL
EDITREC --          /SETTOPVAL
```

[1] The user directs that the functions on le RECORD be analyzed. The leading period and space specify that this line is a Masterscope command. [1]

[2] Masterscope prints a . whenever it (re)analyzes a function, to let the user know what it is happening. [2]

[3] The user asks which functions call RECFIELDLOOK. Masterscope responds with the list.

[4] The user asks to edit the expressions where the function RECFIELDLOOK is called. Masterscope calls EDITF on the functions it had analyzed that call RECFIELDLOOK, directing the editor to the appropriate expressions. The user then edits some of those expressions. [3]

[5] Next the user asks which functions call ERROR. Since some of the functions in the database have been changed, Masterscope re-analyzes the changed de nitions (and prints out .'s for each function it analyzes). Masterscope responds that EDITREC is the only analyzed function that calls ERROR.

[6] The user asks to see a map of the ways in which RECFIELDLOOK is called from ACCESSDEF. A tree structure of the calls is displayed.

---

[1] The user may also call Masterscope directly by typing (MASTERSCOPE). Masterscope prints a greeting and prompts with "_. ". Within the top- level executive of Masterscope, the user may issue Masterscope commands, programmer's assistant commands, (e.g., REDO, FIX), or run programs. The user can exit from the Masterscope executive by typing OK. The function . is de ned as a nlambda nospread function which interprets its argument as a Masterscope command, executes the command and returns.

[2] The feedback when Masterscope analyzes a function is controlled by the ag MSPRINTFLG: if MSPRINTFLG is the atom ".", Masterscope will print out a period. (If an error in the function is detected, "?" is printed instead.) If MSPRINTFLG is a number N, Masterscope will print the name of the function it is analyzing every Nth function. If MSPRINTFLG is NIL, Masterscope won't print anything. Initial setting is ".". Note that the function name is printed when Masterscope starts analyzing, and the comma is printed when it nishes.

[3] In this example, the teletype editor is used. In Interlisp- D, if Dedit is enabled as the primary editor, it would be called to edit the appropriate functions (see page 20.1).

MASTERSCOPE

[7] The user then asks to see which functions call which functions in the list `/FNS`. Masterscope responds with a structured printout of these relations.

Below is a summary of the Masterscope commands, similar to what would be printed out by the `HELP` command (page 13.7). Optional elements are shown in brackets `[]`; alternatives are shown in braces `{}` separated with vertical bars `|` or are listed on separate lines; words in angle brackets `<>` are ''meta-objects''; other lower-case words are ''noise words'' and may be omitted.

```
*----------------------------------------------------------------*
 a <command> is:

   [RE]ANALYZE <functions>
   ERASE <functions>
   show PATHS <pathoptions>
   <set> {<relation> | IS | ARE} <set>
   EDIT where <functions> [<relation> <set>] [ - <edit commands>]
   SHOW where <functions> <relation> <set>
   CHECK <files>
   FOR <variable> <set> <iterative statement tail>
------------------------------------------------------------------
 a <set> is (at least one of):
 a determiner   + a type             + a specification

  THE            FUNCTIONS            ['']{atom | list}
  ANY            VARIABLES            @ <predicate>
  WHICH          PROPERTY NAMES       IN <expression>
  WHO            RECORDS              <relation>ING <set>
                 FIELDS               <relation>ED {BY | IN} <set>
                 FILES                THAT <relation> <set>
                 I.S.OPRS             LIKE <edit-pattern>
                                      ON <files>
                                      ON PATH <pathoptions>
                 FIELDS OF <records>
                 <blockword> {ON <files> | OF <functions>}
  <functions>, <files>, etc. are <set>s whose type is implied.
------------------------------------------------------------------
 a <relation> is a verb and optional modifier:
  verbs:            modifiers (anywhere after the verb):
   CALL             {SOMEHOW | FOR EFFECT | FOR VALUE |
                                    DIRECTLY | INDIRECTLY}
   USE              AS a {RECORD | PROPERTY | record FIELD} name
   USE              AS a CLISP word
   USE              {FREELY | LOCALLY}
   SET              {FREELY | LOCALLY}
   SMASH            {FREELY | LOCALLY}
   TEST             {FREELY | LOCALLY}
   REFERENCE        {FREELY | LOCALLY}
   DECLARE          AS a {LOCALVAR | SPECVAR}
   BIND
   FETCH
   REPLACE      *----------------------------------------------------
```

```
    CREATE        | <blockword>:  ENTRIES, GLOBALVARS, FREEVARS,
    CONTAIN       |  SPECVARS, LOCALFREEVARS, BLKFNS or BLOCKFNS
-------------------------------------------------------------------
  <pathoptions>:               | abbreviations & synonyms:
                               |
    FROM <functions>           | FNS = FUNCTIONS   PROPS = PROPERTIES
    TO <functions>             | VARS = VARIABLES
    AVOIDING <functions>       | (& singular FN, VARIABLE, etc)
    NOTRACE <functions>        | FREE = FREELY    LOCAL = LOCALLY
    SEPARATE <functions>       | AMONG = AVOIDING NOT
    LINELENGTH <number>        |
-------------------------------------------------------------------
    <sets> may be joined by AND or OR or preceded by NOT.
    Any command can be followed by OUTPUT <filename>.
*------------------------------------------------------------------*
```

13.1   COMMAND LANGUAGE

The user communicates with Masterscope using an English-like command language, e.g., WHO CALLS PRINT. With these commands, the user can direct that functions be analyzed, interrogate Masterscope's database, and perform other operations. The commands deal with sets of functions, variables, etc., and relations between them (e.g., call, bind). Sets correspond to English nouns, relations to verbs.

A set of atoms can be specied in a variety of ways, either *explicitly*, e.g., FUNCTIONS ON FIE species the atoms in (FILEFNSLST 'FIE), or *implicitly*, e.g., NOT CALLING Y, where the meaning must be determined in the context of the rest of the command. Such sets of atoms are the basic building blocks which the command language deals with.

Masterscope also deals with relations *between* sets. For example, the relation CALL relates functions and other functions; the relations BIND and USE FREELY relate functions and variables. These relations are what get stored in the Masterscope database when functions are analyzed. In addition, Masterscope ''knows'' about le package conventions; CONTAIN relates les and various types of objects (functions, variables).

Sets and relations are used (along with a few additional words) to form sentence-like *commands*. For example, the command WHO ON 'FOO USE 'X FREELY will print out the list of functions contained in the le FOO which use the variable X freely. The command EDIT WHERE ANY CALLS 'ERROR will call EDITF on those functions which have previously been analyzed that directly call ERROR, pointing at each successive expression where the call to ERROR actually occurs.

13.1.1   Commands

The normal mode of communication with Masterscope is via ''commands''. These are sentences in the Masterscope command language which direct Masterscope to answer questions or perform various operations. The syntax of Masterscope commands is described below:

ANALYZE SET   [Masterscope Command]

> Analyze the functions in SET (and any functions called by them) and include the information gathered in the database. Masterscope will not re-analyzing a function if it thinks it already has valid information about that function in its database. The user may use the command REANALYZE (below) to force re-analysis.

> Note that whenever a function is referred to in a command as a "subject" of one of the relations, it is automatically analyzed; the user need not give an explicit ANALYZE command. Thus, WHO IN MYFNS CALLS FIE will automatically analyze the functions in MYFNS if they have not already been analyzed.

> Note also that only EXPR de nitions will be analyzed; that is, Masterscope will not analyze compiled code. If there is no in-core de nition for a function (either in the de nition cell or an EXPR property), Masterscope will attempt to read in the de nition from a le. [4] If necessary, the de nition will be DWIMIFYed before analysis.

REANALYZE SET   [Masterscope Command]

> Causes Masterscope to reanalyze the functions in SET (and any functions called by them) even if it thinks it already has valid information in its database. For example, this would be necessary if the user had disabled or subverted the le package, e.g. performed PUTD's to change the de nition of functions.

ERASE SET   [Masterscope Command]

> Erase all information about the functions in SET from the database. ERASE by itself clears the entire database.

SHOW PATHS PATHOPTIONS   [Masterscope Command]

> Displays a tree of function calls. PATHOPTIONS are described on page 13.14.

SET RELATION SET   [Masterscope Command]
SET IS SET   [Masterscope Command]
SET ARE SET   [Masterscope Command]

> This command has the same format as an English sentence with a subject (the rst SET), a verb (the RELA TION or IS or ARE), and an object (the second SET). Any of the SETs within the command may be preceded by the question determiners WHICH or WHO (or just WHO alone). For example, WHICH FUNCTIONS CALL X prints the list of functions that call the function X. RELA TION may be one of the relation words in present tense (CALL, BIND, TEST, SMASH, etc.) or used as a passive (e.g., WHO IS CALLED BY WHO). Other variants are allowed, e.g. WHO DOES X CALL, IS FOO CALLED BY FIE, etc.

> The interpretation of the command depends on the number of question elements present:

---

[4]Files which have been explicitly mentioned previously in some command are searched rst. If the de nition cannot be found on any of those les, Masterscope looks among the les on FILELST for a de nition. If a function is found in this manner, Masterscope will print a message "(reading from FILENAME)". If no de nition can be found at all, Masterscope will print a message "FN can't be analyzed". If the function previously was known, the message "FN disappeared! " is printed.

Commands

(1) If there is *no* question element, the command is treated as an assertion and Masterscope returns either `T` or `NIL`, depending on whether that assertion is true. Thus, `ANY IN MYFNS CALL HELP` will print `T` if any function in `MYFNS` call the function `HELP`, and `NIL` otherwise.

(2) If there is *one* question element, Masterscope returns the list of items for which the assertion would be true. For example `MYFN BINDS WHO USED FREELY BY YOURFN` prints the list of variables bound by `MYFN` which are also used freely by `YOURFN`.

(3) If there are two question elements, Masterscope will print a doubly indexed list:

```
_. WHO CALLS WHO IN /FNS cr
RECORDSTATEMENT --  /RPLNODE
RECORDECL1 --       /NCONC, /RPLACD, /RPLNODE
RECREDECLARE1 --    /PUTHASH
UNCLISPTRAN --      /PUTHASH, /RPLNODE2
RECORDWORD --       /RPLACA
RECORD1 --          /RPLACA, /SETTOPVAL
EDITREC --          /SETTOPVAL
```

EDIT WHERE SET RELA TION SET [- EDITCOMS]   [Masterscope Command]
(WHERE may be omitted.) The rst SET refers to a set of functions. The `EDIT` command calls the editor on each expression where the RELA TION actually occurs. For example, `EDIT WHERE ANY CALL ERROR` will call `EDITF` on each (analyzed) function which calls `ERROR` stopping within a `TTY:` at each call to `ERROR`. Currently one cannot `EDIT WHERE` a le which `CONTAINS` a datum, nor where one function `CALLS` another `SOMEHOW`.

EDITCOMS , if given, are a list of commands passed to `EDITF` to be performed at each expression. For example, `EDIT WHERE ANY CALLS MYFN DIRECTLY - (SW 2 3) P` will switch the rst and second arguments to `MYFN` in every call to `MYFN` and print the result. `EDIT WHERE ANY ON MYFILE CALL ANY NOT @ GETD` will call the editor on any expression involving a call to an unde ned function. Note that `EDIT WHERE X SETS Y` will point only at those expressions where `Y` is actually set, and will skip over places where `Y` is otherwise mentioned.

SHOW WHERE SET RELA TION SET   [Masterscope Command]
Like the `EDIT` command except merely prints out the expressions without calling the editor.

EDIT SET [- EDITCOMS]   [Masterscope Command]
Calls `EDITF` on each function in SET. EDITCOMS , if given, will be passed as a list of editor commands to be executed. For example `EDIT ANY CALLING FN1 - (R FN1 FN2)` will replace `FN1` by `FN2` in those functions that call `FN1`.

DESCRIBE SET   [Masterscope Command]
Prints out the `BIND`, `USE FREELY` and `CALL` information about the functions in SET . For example, the command `DESCRIBE PRINTARGS` might print out:

```
PRINTARGS[N,FLG]
```

```
binds:        TEM,LST,X
calls:        MSRECORDFILE,SPACES,PRIN1
called by:    PRINTSENTENCE,MSHELP,CHECKER
```

This shows that `PRINTARGS` has two arguments, `N` and `FLG`, binds internally the variables `TEM`, `LST` and `X`, calls `MSRECORDFILE`, `SPACES` and `PRIN1` and is called by `PRINTSENTENCE`, `MSHELP`, and `CHECKER`.

The user can specify additional information to be included in the description. `DESCRIBELST` is a list each of whose elements is a list containing a descriptive string and a form. The form is evaluated (it can refer to the name of the funtion being described by the free variable `FN`); if it returns a non-`NIL` value, the description string is printed followed by the value. If the value is a list, its elements are printed with commas between them. For example, the entry (`"types: "` `(GETRELATION FN '(USE TYPE) T)` would include a listing of the types used by each function.

CHECK SET   [Masterscope Command]
Checks for various anomolous conditions (mainly in the compiler declarations) for the les in SET (if SET is not given, FILELST is used). For example, this command will warn about variables which are bound but never referenced, functions in BLOCKS delarations which aren't on the le containing the declaration, functions declared as ENTRIES but not in the block, variables which may not need to be declared SPECVARS because they are not used freely below the places where they are bound, etc.

FOR VARIABLE SET I.S.ÆIL   [Masterscope Command]
This command provides a way of combining CLISP iterative statements with Masterscope. An iterative statement will be constructed in which VARIABLE is iteratively assigned to each element of SET, and then the iterative statement tail I.S.ÆIL is executed. For example,

FOR X CALLED BY FOO WHEN CCODEP DO (PRINTOUT T X ,,, (ARGLIST X) T)

will print out the name and argument list of all of the compiled functions which are called by FOO.

HELP   [Masterscope Command]
Prints out a summary of Masterscope commands as shown on page 13.3. Optional elements are shown in brackets [ ]; alternatives are shown in braces {} separated with vertical bars | or are listed on separate lines; words in angle brackets <> are ''meta-objects''; other lower-case words are ''noise words'' and may be omitted.

Note: any command may be followed by OUTPUT FILENAME to send output to the given le rather than the terminal, e.g. WHO CALLS WHO OUTPUT CROSSREF.

13.1.2   Relations

A relation is specied by one of the keywords below. Some of these ''verbs'' accept modiers. For

example, `USE`, `SET`, `SMASH` and `REFERENCE` all may be modi ed by `FREELY`. The modi er may occur anywhere within the command. [5] Verbs can occur in the present tense (e.g., `USE`, `CALLS`, `BINDS`, `USES`) or as present or past participles (e.g., `CALLING`, `BOUND`, `TESTED`). The relations (with their modi ers) recognized by Masterscope are:

`CALL`   [Masterscope Relation]
> Function `F1` calls `F2` if the de nition of `F1` contains a form `(F2 --)`, `(APPLY (QUOTE F2) --)`, `(FUNCTION F2)`, etc.

`CALL SOMEHOW`   [Masterscope Relation]
> One function calls another `SOMEHOW` if there is some path from the rst to the other. That is, if `F1` calls `F2`, and `F2` calls `F3`, then `F1` `CALLS` `F3` `SOMEHOW`.
>
> This information is not stored directly in the database; instead, Masterscope stores only information about direct function calls, and (re)computes the `CALL SOMEHOW` relation as necessary.

`USE`   [Masterscope Relation]
> If unmodi ed, the relation `USE` denotes variable usage in any way; it is the union of the relations `SET`, `SMASH`, `TEST`, and `REFERENCE`.

`SET`   [Masterscope Relation]
> A function `SET`s a variable if the function contains a form `(SETQ var --)`, `(SETQQ var --)`, etc.

`SMASH`   [Masterscope Relation]
> A function `SMASH`es a variable if the function calls a destructive list operation (`RPLACA`, `RPLACD`, `DREMOVE`, `SORT`, etc.)   on the value of that variable. Masterscope will also nd instances where the operation is performed on a ''part'' of the value of the variable; for example, if a function contains a form `(RPLACA (NTH X 3) T)` it will be noted as `SMASHING` `X`.
>
> Note that if the function contains a sequence `(SETQ Y X)`, `(RPLACA Y T)` then `Y` is noted as being smashed, but not `X`.

`TEST`   [Masterscope Relation]
> A variable is `TEST`ed by a function if its value is only distinguished between `NIL` and non-`NIL`. For example, the form `(COND ((AND X --) --))` *tests* the value of `X`.

`REFERENCE`   [Masterscope Relation]
> This relation includes all variable usage *except* for `SET`.

The verbs `USE`, `SET`, `SMASH`, `TEST` and `REFERENCE` may be modi ed by the words `FREELY` or `LOCALLY`. A variable is used `FREELY` if it is not bound in the function at the place of its use; alternatively, it is used `LOCALLY` if the use occurs within a `PROG` or `LAMBDA` that binds the variable.

---

[5]If there is more than one verb, any modi er *between* two verbs is assumed to modify the rst one. For example, in `USING ANY FREELY OR SETTING X`, the `FREELY` modi es `USING` but not `SETTING` the entire phrase is interpreted as the set of all functions which either use any variable freely or set the variable `X`, whether or not `X` is set freely.

MASTERSCOPE

Masterscope also distinguishes between CALL DIRECTLY and CALL INDIRECTLY. A function is called DIRECTLY if it occurs as CAR-of-form in a normal evaluation context. A function is called INDIRECTLY if its name appears in a context which does not imply its *immediate* evaluation, for example (SETQ Y (LIST (FUNCTION FOO) 3)).[6] In addition, CALL FOR EFFECT (where the value of the function is not used) is distinguished from CALL FOR VALUE.

BIND   [Masterscope Relation]
> The BIND relation between functions and variables includes both variables bound as function arguments and those bound in an internal PROG or LAMBDA expression.

USE AS A FIELD   [Masterscope Relation]
> Masterscope notes all uses of record eld names within FETCH, REPLACE or CREATE expressions.

FETCH   [Masterscope Relation]
> Use of a eld within a FETCH expression.

REPLACE   [Masterscope Relation]
> Use of a record eld name within a REPLACE or CREATE expression.

USE AS A RECORD   [Masterscope Relation]
> Masterscope notes all uses of record names within CREATE or TYPE? expressions.[7]

CREATE   [Masterscope Relation]
> Use of a record name within a CREATE expression.

USE AS A PROPERTY NAME   [Masterscope Relation]
> Masterscope notes the property names used in GETPROP, PUTPROP, GETLIS, etc. expressions if the name is quoted. E.g. if a function contains a form (GETPROP X (QUOTE INTERP)), then that function USEs INTERP as a property name.

USE AS A CLISP WORD   [Masterscope Relation]
> Masterscope notes all iterative statement operators and user de ned CLISP words as being used as a CLISP word.

CONTAIN   [Masterscope Relation]
> Files *contain* functions, records, and variables. This relation is not stored in the database but is computed using the le package.

DECLARE AS LOCALVAR   [Masterscope Relation]
DECLARE AS SPECVAR   [Masterscope Relation]
> Masterscope notes internal ''calls'' to DECLARE from within functions.

The following abbreviations are recognized: FREE= FREELY, LOCAL= LOCALLY, PROP= PROPERTY, REF= REFERENCE. Also, the words A, AN and NAME (after AS) are ''noise'' words and may be omitted.

---

[6]The distinction is whether or not the compiled code of the caller would contain a direct call to the callee. Note that an occurrence of (FUNCTION FOO) as the functional argument to one of the built- in mapping functions which compile open is considered to be a direct call.

[7]Additionally, in X:FOO.FIE, FOO is used as a record name.

Note: Masterscope uses ''templates'' (page 13.16) to decide which relations hold between functions and their arguments. For example, the information that SORT SMASHes its rst argument is contained in the template for SORT. Masterscope initially contains templates for most system functions which set variables, test their arguments, or perform destructive operations. The user may change existing templates or insert new ones in Masterscope's tables via the SETTEMPLATE function (page 13.19).

13.1.3   Sets

A ''set'' is a collection of things (functions, variables, etc.). A set is speci ed by a set phrase, consisting of a *determiner* (e.g., ANY, WHICH, WHO) followed by a *type* (e.g., FUNCTIONS, VARIABLES) followed by a *speci cation* (e.g., IN MYFNS, @ SUBRP). The determiner, type and speci cation may be used alone or in combination. For example, ANY FUNCTIONS IN MYFNS, ANY @ SUBRP, VARIABLES IN GLOBALVARS, and WHO are all acceptable set phrases. Set speci cations, types and determiners are explained below:

13.1.3.1   Set Speci cations

'ATOM   [Masterscope Set Speci cation]
>The simplest way to specify a set consisting of a single thing is by the name of that thing. For example, in the command WHO CALLS 'ERROR, the function ERROR is referred to by its name. Although the ' can be left out, to resolve possible ambiguities names should usually be quoted; e.g., WHO CALLS 'CALLS will return the list of functions which call the function CALLS.

'LIST   [Masterscope Set Speci cation]
>Sets consisting of several atoms may be speci ed by naming the atoms. For example, the command WHO USES '(A B) returns the list of functions that use the variables A or B.

IN EXPRESSION   [Masterscope Set Speci cation]
>The form EXPRESSION is evaluated, and its value is treated as a list of the elements of a set. For example, IN GLOBALVARS speci es the list of variables in the value of the variable GLOBALVARS.

@ PREDICA TE   [Masterscope Set Speci cation]
>A set may also be speci ed by giving a predicate which the elements of that set must satisfy. PREDICA TE is either a function name, a LAMBDA expression, or an expression in terms of the variable X. The speci cation @ PREDICA TE represents all atom for which the value of PREDICA TE is non-NIL. For example, @ EXPRP speci es all those atoms which have EXPR de ntions; @ (STRPOSL X CLISPCHARRAY) speci es those atoms which contain CLISP characters. The universe to be searched is either determined by the context within the command (e.g., in WHO IN FOOFNS CALLS ANY NOT @ GETD, the predicate is only applied to functions which are called by any functions in the list FOOFNS), or in the extreme case, the universe defaults to the entire set of things which have been noticed by Masterscope, as in the command WHO IS @ EXPRP.

LIKE ATOM   [Masterscope Set Speci cation]
>ATOM may contain ESCs; it is used as a pattern to be matched (as in the editor).

MASTERSCOPE

For example, `WHO LIKE /R$ IS CALLED BY ANY` would nd both `/RPLACA` and `/RPLNODE`.

A set may also be speci ed by giving a relation its members must have with the members of another set:

`RELA TION ING SET`   [Masterscope Set Speci cation]

> `RELA TION ING` is used here generically to mean any of the relation words in the present participle form (possibly with a modi er), e.g., `USING`, `SETTING`, `CALLING`, `BINDING`. `RELA TION ING SET` speci es the set of all objects which have that relation with some element of `SET`. For example, `CALLING X` speci es the set of functions which call the function `X`; `USING ANY IN FOOVARS FREELY` speci es the set of functions which uses freely any variable in the value of `FOOVARS`.

`RELA TION ED BY SET`   [Masterscope Set Speci cation]
`RELA TION ED IN SET`   [Masterscope Set Speci cation]

> This is similar to the `RELA TION ING` construction. For example, `CALLED BY ANY IN FOOFNS` represents the set of functions which are called by any element of `FOOFNS`; `USED FREELY BY ANY CALLING ERROR` is the set of variables which are used freely by any function which also calls the function `ERROR`.

`BLOCKTYPE  OF FUNCTIONS`   [Masterscope Set Speci cation]
`BLOCKTYPE  ON FILES`   [Masterscope Set Speci cation]

> These phrases allow the user to ask about `BLOCKS` declarations on les (see page 12.14). `BLOCKTYPE` is one of `LOCALVARS`, `SPECVARS`, `GLOBALVARS`, `ENTRIES`, `BLKFNS`, `BLKAPPLYFNS`, or `RETFNS`.

> `BLOCKTYPE  OF FUNCTIONS` speci es the names which are declared to be `BLOCKTYPE` in any blocks declaration which contain any of `FUNCTIONS` (a ''set'' of functions). The ''functions'' in `FUNCTIONS` can either be block names or just functions in a block. For example, `WHICH ENTRIES OF ANY CALLING 'Y BIND ANY GLOBALVARS ON 'FOO`.

> `BLOCKTYPE  ON FILES` speci es all names which are declared to be `BLOCKTYPE` on any of the given `FILES` (a ''set'' of les).

`FIELDS OF SET`   [Masterscope Set Speci cation]

> `SET` is a set of records. This denotes the eld names of those records. For example, the command `WHO USES ANY FIELDS OF BRECORD` returns the list of all functions which do a `fetch` or `replace` with any of the eld names declared in the record declaration of `BRECORD`.

`KNOWN`   [Masterscope Set Speci cation]

> The set of all functions which have been analyzed. For example, the command `WHO IS KNOWN` will print out the list of functions which have been analyzed.

`THOSE`   [Masterscope Set Speci cation]

> The set of things printed out by the last Masterscope question. For example, following the command `WHO IS USED FREELY BY PARSE`, the user could ask `WHO BINDS THOSE` to nd out where those variables are bound.

`ON PATH PATHOPTIONS`   [Masterscope Set Speci cation]

> Refers to the set of functions which *would be* printed by the command `SHOW PATHS`

13.11

PATHOPTIONS . For example, `IS FOO BOUND BY ANY ON PATH TO 'PARSE` tests if FOO might be bound ''above'' the function `PARSE`. PATHOPTIONS are explained in detail on page 13.14.

Note: sets may also be speci ed with ''relative clauses'' introduced by the word THAT, e.g. `THE FUNCTIONS THAT BIND 'X`.

13.1.3.2 Set Determiners

Set phrases may be preceded by a *determiner*. A determiner is one of the words THE, ANY, WHO or WHICH. The ''question'' determiners (WHO and WHICH) are only meaningful in some of the commands, namely those that take the form of questions. ANY and WHO (or WHOM) can be used alone; they are ''wild-card'' elements, e.g., the command `WHO USES ANY FREELY`, will print out the names of all (known) functions which use any variable freely. If the determiner is omitted, ANY is assumed; e.g. the command `WHO CALLS '(PRINT PRIN1 PRIN2)` will print the list of functions which call *any* of PRINT, PRIN1, PRIN2. THE is also allowed, e.g. `WHO USES THE RECORD FIELD FIELDX`.

13.1.3.3 Set Types

Any set phrase has a *type*; that is, a set may specify either functions, variables, les, record names, record eld names or property names. The type may be determined by the context within the command (e.g., in `CALLED BY ANY ON FOO`, the set `ANY ON FOO` is interpreted as meaning the *functions* on FOO since only functions can be CALLED), or the type may be given explicitly by the user (e.g., `FUNCTIONS ON FIE`). The following types are recognized: FUNCTIONS, VARIABLES, FILES, PROPERTY NAMES, RECORDS, FIELDS, I.S.OPRS.[8]

The type is used by Masterscope in a variety of ways when interpreting the set phrase:

(1) Set types are used to disambiguate possible parsings. For example, both commands `WHO SETS ANY BOUND IN X OR USED BY Y` and `WHO SETS ANY BOUND IN X OR CALLED BY Y` have the same general form. However, the rst case is parsed as `WHO SETS ANY (BOUND BY X OR USED BY Y)` since both `BOUND BY X` and `USED BY Y` refer to variables; while the second case as `WHO SETS ANY BOUND IN (X OR CALLED BY Y)`, since `CALLED BY Y` and X must refer to functions. Note that parentheses may be used to group phrases.

(2) The type is used to determine the modi er for USE: `FOO USES WHICH RECORDS` is equivalent to `FOO USES WHO AS A RECORD FIELD`.

(3) The interpretation of CONTAIN depends on the type of its object: the command `WHAT FUNCTIONS ARE CONTAINED IN MYFILE` prints the list of functions in MYFILE; `WHAT RECORDS ARE ON MYFILE` prints the list of records.

(4) The implicit ''universe'' in which a set expression is interpreted depends on the type: `ANY VARIABLES @ GETD` is interpreted as the set of all variables which have been noticed by Masterscope (i.e., bound or

---

[8]or abbreviations FNS, VARS, PROPNAMES or the singular forms FUNCTION, FN, VARIABLE, VAR, FILE, PROPNAME, RECORD, FIELD. Note that most of these types correspond to built-in '' le package types'' (see page 11.14).

used in any function which has been analyzed) that also have a de nition. ANY FUNCTIONS @ (NEQ (GETTOPVAL X) 'NOBIND) is interpreted as the set of all functions which have been noticed (either analyzed or called by a function which has been analyzed) that also have a top- level value.

13.1.4   Conjunctions

Sets may be joined by the conjunctions AND and OR or preceded by NOT to form new sets. AND is always interpreted as meaning ''intersection''; OR as ''union'', while NOT means ''complement''. For example, the set CALLING X AND NOT CALLED BY Y speci es the set of all functions which call the function X but are not called by Y.

Masterscope's interpretation of AND and OR follow LISP conventions rather than the conventional English interpretation. For example ''calling X and Y'' would, in English, be interpreted as the intersection of (CALLING X) and (CALLING Y); but Masterscope interprets CALLING X AND Y as CALLING ('X AND 'Y); which is the null set. Only sets may be joined with conjunctions: joining modi ers, as in USING X AS A RECORD FIELD OR PROPERTY NAME, is not allowed; in this case, the user must say USING X AS A RECORD FIELD OR USING X AS A PROPERTY NAME.

As described above, the type of sets is used to disambiguate parsings. The algorithm used is to rst try to match the type of the phrases being joined and then try to join with the longest preceding phrase. In any case, the user may group phrases with parentheses to specify the manner in which conjunctions should be parsed.

13.2   PATHS

In trying to work with large programs, the user can lose track of the hierarchy of functions. The Masterscope SHOW PATHS command aids the user by providing a map showing the calling structure of a set of functions. SHOW PATHS prints out a tree structure showing which functions call which other functions. For example, the command SHOW PATHS FROM MSPARSE will print out the structure of Masterscope's parser:

```
1.MSPARSE   MSINIT MSMARKINVALID
2.          |       MSINITH MSINITH
3.          MSINTERPRET MSRECORDFILE
4.          |           MSPRINTWORDS
5.          |           PARSECOMMAND GETNEXTWORD CHECKADV
6.          |           |            PARSERELATION {a}
7.          |           |            PARSESET {b}
8.          |           |            PARSEOPTIONS {c}
9.          |           |            MERGECONJ GETNEXTWORD {5}
10.         |           GETNEXTWORD {5}
11.         |           FIXUPTYPES SUBJTYPE
12.         |           |          OBJTYPE
13.         |           FIXUPCONJUNCTIONS MERGECONJ {9}
14.         |                             MATCHSCORE
15.         MSPRINTSENTENCE
----------------------------------------------------- overflow - a
```

```
16.PARSERELATION GETNEXTWORD {5}
17.              CHECKADV
-------------------------------------------------- overflow - b
19.PARSESET PARSESET
20.         GETNEXTWORD {5}
21.         PARSERELATION {6}
22.         SUBPARSE GETNEXTWORD {5}
-------------------------------------------------- overflow - c
23.PARSEOPTIONS GETNEXTWORD {5}
24.              PARSESET {19}
```

The above printout displays that the function MSPARSE calls MSINIT, MSINTERPRET, and MSPRINTSENTENCE. MSINTERPRET in turn calls MSRECORDFILE, MSPRINTWORDS, PARSECOMMAND, GETNEXTWORD, FIXUPTYPES, and FIXUPCONJUNCTIONS. The numbers in braces {} after a function name are backward references: they indicate that the tree for that function was expanded on a previous line. The lowercase letters in braces are *forward* references: they indicate that the tree for that function will be expanded below, since there is no more room on the line. The vertical bar is used to keep the output aligned.

Note: In Interlisp-D, the Browser Lispusers package modi es the SHOW PATHS command so the command's output is displayed as an undirected graph (see page 18.9).

13.2.1   Path Options

The SHOW PATHS command takes the form: SHOW PATHS followed by some combination of the following *path options:*

FROM SET   [Masterscope Path Option]
> Display the function calls from the elements of SET.

TO SET   [Masterscope Path Option]
> Display the function calls leading *to* elements of SET. If TO is given before FROM (or no FROM is given), the tree is ''inverted'' and a message, (inverted tree) is printed to warn the user that if FN1 appears after FN2 it is because FN1 is *called by* FN2.

When both FROM and TO are given, the rst one indicates a set of functions which are to be displayed while the second restricts the paths that will be traced; i.e., the command SHOW PATHS FROM X TO Y will trace the elements of the set CALLED SOMEHOW BY X AND CALLING Y SOMEHOW.

If TO is not given, TO KNOWN OR NOT @ GETD is assumed; that is, only functions which have been analyzed or which are unde ned will be included. Note that Masterscope will analyze a function while printing out the tree if that function has not previously been seen and it currently has an EXPR de nition; thus, any function which *can be* analyzed will be displayed.

AVOIDING SET   [Masterscope Path Option]
> Do not display any function in SET. AMONG is recognized as a synonym for AVOIDING NOT. For example, SHOW PATHS TO ERROR AVOIDING ON FILE2 will not display (or trace) any function on FILE2.

NOTRACE SET   [Masterscope Path Option]

Do not trace from any element of SET. NOTRACE di ers from AVOIDING in that a function which is marked NOTRACE will be printed, but the tree beyond it will not be expanded; the functions in an AVOIDING set will not be printed at all. For example, SHOW PATHS FROM ANY ON FILE1 NOTRACE ON FILE2 will display the tree of calls eminating from FILE1, but will not expand any function on FILE2.

SEPARATE SET   [Masterscope Path Option]

Give each element of SET a separate tree. Note that FROM and TO only insure that the designated functions will be displayed. SEPARATE can be used to guarantee that certain functions will begin new tree structures. SEPARATE functions are displayed in the same manner as over ow lines; i.e., when one of the functions indicated by SEPARATE is found, it is printed followed by a forward reference (a lower-case letter in braces) and the tree for that function is then expanded below.

LINELENGTH N   [Masterscope Path Option]

Resets LINELENGTH to N before displaying the tree. The linelength is used to determine when a part of the tree should ''over ow'' and be expanded lower.

13.3   ERROR MESSAGES

When the user gives Masterscope a command, the command is rst parsed, i.e. translated to an internal representation, and then the internal representation is interpreted. If a command cannot be parsed, e.g. if the user typed SHOW WHERE CALLED BY X, the message ''Sorry, I can't parse that!'' is printed and an error is generated. If the command is of the correct form but cannot be interpreted (e.g., the command EDIT WHERE ANY CONTAINS ANY) Masterscope will print the message 'Sorry, that isn't implemented!'' and generate an error. If the command requires that some functions having been analyzed (e.g., the command WHO CALLS X) and the database is empty, Masterscope will print the message ''Sorry, no functions have been analyzed!'' and generate an error.

13.4   MACRO EXPANSION

As part of analysis, Masterscope will expand the macro de nition of called functions, if they are not otherwise de ned (see page 5.17). Masterscope macro expansion is controlled by the variable MSMACROPROPS:

MSMACROPROPS   [Variable]

Value is an ordered list of macro-property names that Masterscope will search to nd a macro de nition. Only the kinds of macros that appear on MSMACROPROPS will be expanded. All others will be treated as function calls and left unexpanded.

Initially (MACRO).

Note: MSMACROPROPS initially contains only MACRO (and not 10MACRO, DMACRO,

etc.) in the theory that the machine-dependent macro de nitions are more likely ''optimizers''.

Note that if you edit a macro, Masterscope will know to reanalyze the functions which call that macro. However, if your macro is of the ''computed-macro'' style, and it calls functions which you edit, Masterscope will not notice. You must be careful to tell masterscope to REANALYZE the appropriate functions (e.g., if you edit FOOEXPANDER which is used to expand FOO macros, you have to . REANALYZE ANY CALLING FOO.

13.5   AFFECTING MASTERSCOPE ANALYSIS

Masterscope analyzes the EXPR de nitions of functions and notes in its database the relations that function has with other functions and with variables. To perform this analysis, Masterscope uses *templates* which describe the behavior of functions. For example, the information that SORT SMASHes its rst argument is contained in the template for SORT. Masterscope initially contains templates for most system functions which set variables, test their arguments, or perform destructive operations.

A template is a list structure containing any of the following atoms:

PPE   [in Masterscope template]
> If an expression appears in this location, there is most likely a parenthesis error.
>
> Masterscope notes this as a ''call'' to the function ''ppe'' (lowercase). Therefore, SHOW WHERE ANY CALLS ppe will print out all possible parenthesis errors. When Masterscope nds a possible parenthesis error in the course of analyzing a function de nition, rather than printing the usual ''.'', it prints out a ''?'' instead.

NIL   [in Masterscope template]
> The expression occuring at this location is not evaluated.

SET   [in Masterscope template]
> A variable appearing at this place is set.

SMASH   [in Masterscope template]
> The value of this expression is smashed.

TEST   [in Masterscope template]
> This expression is used as a predicate (that is, the only use of the value of the expression is whether it is NIL or non-NIL).

PROP   [in Masterscope template]
> The value of this expression is used as a property name. If the expression is of the form (QUOTE ATOM), Masterscope will note that ATOM is USED AS A PROPERTY NAME. For example, the template for GETPROP is (EVAL PROP . PPE).

FUNCTION   [in Masterscope template]
> The expression at this point is used as a functional argument. For example, the template for MAPC is (SMASH FUNCTION FUNCTION . PPE).

FUNCTIONAL   [in Masterscope template]
> The expression at this point is used as a functional argument. This is like
> FUNCTION, except that Masterscope distinguishes between functional arguments to
> functions which ''compile open'' from those that do not. For the latter (e.g. SORT
> and APPLY), FUNCTIONAL should be used rather than FUNCTION.

EVAL   [in Masterscope template]
> The expression at this location is evaluated (but not set, smashed, tested, used as a
> functional argument, etc.).

RETURN   [in Masterscope template]
> The value of the function (of which this is the template) is the value of this
> expression.

TESTRETURN   [in Masterscope template]
> A combination of TEST and RETURN: If the value of the function is non-NIL,
> then it is returned. For instance, a one-element COND clause is this way.

EFFECT   [in Masterscope template]
> The expression at this location is evaluated, but the value is not used.

FETCH   [in Masterscope template]
> An atom at this location is a eld which is fetched.

REPLACE   [in Masterscope template]
> An atom at this location is a eld which is replaced.

RECORD   [in Masterscope template]
> An atom at this location is used as a record name.

CREATE   [in Masterscope template]
> An atom at this location is a record which is created.

BIND   [in Masterscope template]
> An atom at this location is a variable which is bound.

CALL   [in Masterscope template]
> An atom at this location is a function which is called.

CLISP   [in Masterscope template]
> An atom at this location is used as a CLISP word.

!   [in Masterscope template]
> This atom, which can only occur as the rst element of a template, allows one to
> specify a template for the CAR of the function form. If ! doesn't appear, the CAR
> of the form is treated as if it had a CALL speci ed for it. In other words, the
> templates (.. EVAL) and (! CALL .. EVAL) are equivalent.
>
> If the next atom after a ! is NIL, this speci es that the function name should
> not be remembered. For example, the template for AND is (! NIL .. TEST
> RETURN), which means that if you see an ''AND'', don't remember it as being
> called. This keeps the Masterscope database from being cluttered by too many
> uninteresting relations; Masterscope also throws away relations for COND, CAR,

CDR, and a couple of others.

In addition to the above atoms which occur in templates, there are some ''special forms'' which are lists keyed by their CAR.

`..    TEMPLA TE`   [in Masterscope template]
>   Any part of a template may be preceded by the atom `..` (two periods) which speci es that the template should be repeated an inde nite number (N   0) of times to ll out the expression. For example, the template for COND might be `(.. (TEST .. EFFECT RETURN))` while the template for SELECTQ is `(EVAL .. (NIL .. EFFECT RETURN) RETURN)`.

`(BOTH TEMPLA TE`$_1$` TEMPLA TE`$_2$`)`   [in Masterscope template]
>   Analyze the current expression twice, using the each of the templates in turn.

`(IF EXPRESSION   TEMPLA TE`$_1$` TEMPLA TE`$_2$`)`   [in Masterscope template]
>   Evaluate EXPRESSION   at analysis time (the variable EXPR will be bound to the expression which corresponds to the IF), and if the result is non- NIL, use TEMPLA TE$_1$, otherwise TEMPLA TE$_2$. If EXPRESSION   is a literal atom, it is APPLY'd to EXPR. For example, `(IF LISTP (RECORD FETCH) FETCH)` speci es that if the current expression is a list, then the rst element is a record name and the second element a eld name, otherwise it is a eld name.

`(@ EXPRF ORM   TEMPLA TEF ORM )`   [in Masterscope template]
>   Evaluate EXPRF ORM   giving EXPR , evaluate TEMPLA TEF ORM   giving TEMPLA TE. Then analyze EXPR   with TEMPLA TE. @ lets the user compute on the y both a template and an expression to analyze it with. The forms can use the variable EXPR, which is bound to the current expression.

`(MACRO . MA CR O )`   [in Masterscope template]
>   MA CR O is interpreted in the same way as a macro (see page 5.17) and the resulting form is analyzed. If the template is the atom MACRO alone, Masterscope will use the MACRO property of the function itself. This is useful when analyzing code which contains calls to user- de ned macros. If the user changes a macro property (e.g. by editing it) of an atom which has template of MACRO, Masterscope will mark any function which used that macro as needing to be reanalyzed.

Some examples of templates:

function	template
DREVERSE	`(SMASH . PPE)`
AND	`(! NIL TEST .. RETURN)`
MAPCAR	`(EVAL FUNCTION FUNCTION)`
COND	`(! NIL .. (IF CDR (TEST .. EFFECT RETURN) (TESTRETURN . PPE)))`

Templates may be changed and new templates de ned using the functions:

`(GETTEMPLATE FN )`   [Function]
>   Returns the current template of FN .

(SETTEMPLATE FN TEMPLA TE) [Function]

> Changes the template for the function FN and returns the old value. If any functions in the database are marked as calling FN , they will be marked as needing re-analysis.

13.6   DATA BASE UPDATING

Masterscope is interfaced to the editor and le package so that it notes whenever a function has been changed, either through editing or loading in a new de nition. Whenever a command is given which requires knowing the information about a speci c function, if that function has been noted as being changed, the function is automatically re-analyzed before the command is interpreted. If the command requires that all the information in the database be consistent (e.g., the user asks WHO CALLS X) then *all* functions which have been marked as changed are re-analyzed.

13.7   MASTERSCOPE ENTRIES

(CALLS FN USED ATABASE _) [Function]

> FN can be a function name, a de nition, or a form. Note: CALLS will also work on compiled code. CALLS returns a list of four elements: a list of all the functions called by FN ,[9] a list of all the variables bound in FN , a list of all the variables used freely in FN , and a list of the variables used globally in FN . For the purpose of CALLS, variables used freely which are on GLOBALVARS or have a property GLOBALVAR value T are considered to be used globally. If USED ATABASE is NIL (or FN is not a litatom), CALLS will perform a one-time analysis of FN . Otherwise (i.e. if USED ATABASE is non-NIL and FN a function name), CALLS will use the information in Masterscope's database (FN will be analyzed rst if necessary).

(CALLSCCODE FN _) [Function]

> The sub-function of CALLS which analyzes compiled code. CALLSCCODE returns a list of *ve* elements: a list of all the functions called via ''linked'' function calls, a list of all functions called regularly, a list of variables bound in FN , a list of variables used freely, and a list of variables used globally.

(FREEVARS FN USED ATABASE) [Function]

> Equivalent to (CADDR (CALLS FN USED ATABASE)). Returns the list of variables used freely within FN .

(MASTERSCOPE COMMAND _) [Function]

> Top level entry to Masterscope. If COMMAND is NIL, will enter into a USEREXEC in which the user may enter commands. If COMMAND is not NIL, the command

---

[9]Functions called via ''linked'' calls from compiled code are indicated by semicolons PACKed around their name; e.g. (CALLS 'MASTERSCOPE) might return ((;MASTERSCOPEEXEC; ;MSINTERPRET; ;PRINT; HELP) --). This feature can be suppressed by setting NOPACKCALLSFLG to T.

is interpreted and MASTERSCOPE will return the value that would be printed by the command. Note that only the question commands return meaningful values.

(SETSYNONYM PHRASE MEANING _)   [Function]

De nes a new synonym for Masterscope's parser. Both PHRASE and MEANING are lists of words; anywhere PHRASE is seen in a command, MEANING will be substituted. For example, (SETSYNONYM 'GLOBALS '(VARS IN GLOBALVARS OR @(GETPROP X 'GLOBALVAR))) would allow the user to refer with the single word GLOBALS to the set of variables which are either in GLOBALVARS or have a GLOBALVAR property.

The following functions are provided for users who wish to write their own routines using Masterscope's database:

(PARSERELATION RELA TION)   [Function]

RELA TION is a relation phrase; e.g., (PARSERELATION '(USE FREELY)). PARSERELATION returns an internal representation for RELA TION. For use in conjunction with GETRELATION.

(GETRELATION ITEM RELA TION INVER TED)   [Function]

RELA TION is an internal representation as returned by PARSERELATION (if not, GETRELATION will rst perform (PARSERELATION RELA TION)); ITEM is an atom. GETRELATION returns the list of all atoms which have the given relation to ITEM. For example, (GETRELATION 'X '(USE FREELY)) returns the list of variables that X uses freely. If INVER TED is T, the inverse relation is used; e.g. (GETRELATION 'X '(USE FREELY) T) returns the list of functions which use X freely.

If ITEM is NIL, GETRELATION will return the list of atoms which have RELA TION with *any* other item; i.e., answers the question WHO RELA TION S ANY. Note that GETRELATION does *not* check to see if ITEM has been analyzed, or that other functions that have been changed have been re-analyzed.

(TESTRELATION ITEM RELA TION ITEM2 INVER TED)   [Function]

equivalent to (MEMB ITEM2 (GETRELATION ITEM RELA TION INVER TED)), that is, tests if ITEM and ITEM2 are related via RELA TION. If ITEM2 is NIL, the call is equiv alent to (NOT (NULL (GETRELATION ITEM RELA TION INVER TED))), i.e., TESTRELATION tests if ITEM has the given RELA TION with *any* other item.

(MAPRELATION RELA TION MAPFN)   [Function]

Calls the function MAPFN on every pair of items related via RELA TION. If (NARGS MAPFN) is 1, then MAPFN is called on every item which has the given RELA TION to *any* other item.

(MSNEEDUNSAVE FNS MSG MARK CHANGEFL G)   [Function]

Used to mark functions which depend on a changed record declaration (or macro, etc.), and which must be LOADed or UNSAVEd (see below). FNS is a list of functions to be marked, and MSG is a string describing the records, macros, etc. on which they depend. If MARK CHANGEFL G is non-NIL, each function in the list is marked as needing re-analysis.

(UPDATEFN FN EVENIFVALID _)   [Function]

> Equivalent to the command ANALYZE 'FN; that is, UPDATEFN will analyze FN if FN has not been analyzed before or if it has been changed since the time it was analyzed. If EVENIFVALID is set, UPDATEFN will re-analyze FN even if Masterscope thinks it has a valid analysis in the database.

(UPDATECHANGED)   [Function]

> Performs (UPDATEFN FN) on every function which has been marked as changed.

(MSMARKCHANGED FN TYPE REASON)   [Function]

> Mark that FN has been changed and needs to be reanalyzed. See MARKASCHANGED, page 11.11.

(DUMPDATABASE FNLST)   [Function]

> Dumps the current Masterscope database on the current output le in a LOADable form. If FNLST is not NIL, DUMPDATABASE will only dump the information for the list of functions in FNLST. The variable DATABASECOMS is initialized to ((E (DUMPDATABASE))); thus, the user may merely perform (MAKEFILE 'DATABASE.EXTENSION) to save the current Masterscope database. If a Masterscope database already exists when a DATABASE le is loaded, the database on the le will be merged with the one in core. Note that functions whose de nitions are di erent from their de nition when the database was made must be REANALYZEd if their new de nitions are to be noticed.
>
> The Databasefns package (page 23.15) provides a more convenient way of saving data bases along with the source les which they correspond to.

13.8   NOTICING CHANGES THAT REQUIRE RECOMPILING

When a record declaration, iterative statement operator or macro is changed, and Masterscope has ''noticed'' a use of that declaration or macro (i.e. it is used by some function known about in the data base), Masterscope will alert the user about those functions which might need to be re-compiled (e.g. they do not currently have EXPR de nitions). [10] The functions which need recompiling are added to the list MSNEEDUNSAVE and a message is printed out:

```
The functions FN1 , FN2 ,... use macros which have changed.
Call UNSAVEFNS() to load and/or unsave them.
```

In this situation, the following function is useful:

(UNSAVEFNS _)   [Function]

> Uses LOADFNS or UNSAVEDEF to make sure that all functions in the list MSNEEDUNSAVE have EXPR de nitions, and then sets MSNEEDUNSAVE to NIL.

---

[10]Extra functions may be noticed; for example if FOO contains (fetch (REC X) --), and some declaration other than REC which contains X is changed, Masterscope will still think that FOO needs to be loaded/unsaved.

13.9   IMPLEMENTATION NOTES

Masterscope keeps a database of the relations noticed when functions are analyzed.   The relations are intersected to form ''primitive relationships'' such that there is little or no overlap of any of the primitives. For example, the relation `SET` is stored as the union of `SET LOCAL` and `SET FREE`. The `BIND` relation is divided into `BIND AS ARG`, `BIND AND NOT USE`, and `SET LOCAL`, `SMASH LOCAL`, etc. Splitting the relations in this manner reduces the size of the database considerably, to the point where it is reasonable to maintain a Masterscope database for a large system of functions during a normal debugging session.

Each primitive relationship is stored in a pair of hash-tables, one for the ''forward'' direction and one for the ''reverse''. For example, there are two hash tables, `USE AS PROPERTY` and `USED AS PROPERTY`. To retrieve the information from the database, Masterscope performs unions of the hash-values.   For example, to answer `FOO BINDS WHO` Masterscope will look in all of the tables which make up the `BIND` relation.   The ''internal representation'' returned by `PARSERELATION` is just a list of dotted pairs of hash-tables. To perform `GETRELATION` requires only mapping down that list, doing `GETHASH`'s on the appropriate hash-tables and `UNION`ing the result.

Hash tables are used for a variety of reasons: storage space is smaller; it is not necessary to maintain separate lists of which functions have been analyzed (a special table, `DOESN'T DO ANYTHING` is maintained for functions which neither call other functions nor bind or use any variables); and accessing is relatively fast. Within any of the tables, if the hash-value would be a list of one atom, then the atom itself, rather than the list, is stored as the hash-value. This also reduces the size of the database signi cantly.