THE METAL PARSING SYSTEM

Jonathan Slocum

Working Paper
February 1981
LRC-81-2

ABSTRACT

A parsing system for defining, debugging, editing and saving collections of grammar rules and lexical entries is described. The system is therefore much more than a mere parser; it is an environment for constructing theories of natural language, at least for the purpose of analysis. It has been used to implement a theory of analysis of one human language for translation into another. The system was intended to provide a means, not just of analysis, but of highly efficient analysis which would make large-scale verification of linguistic theories both possible in theory and affordable in practice. To that end, it has been highly successful.

Introduction

There are actually two METAL parsers: one implements a Left-Corner algorithm, and the other the Cocke-Kasami-Younger bottom-up algorithm. They are imbedded in the identical environment of "help" routines for defining, editing, deleting, and querying the contents of collections of augmented context-free grammar rules and lexical entries. The system supports the maintenance of independent grammars (collections of rules and lexical entries), useful for example when comparing the performance of alternative grammars on a given text corpus. Another important difference between the METAL parsing system and a "plain" parser is the degree of control afforded the user during the parsing process. Most parsers offer little or nothing more than a means of introducing context-free rules and lexical entries, plus some way to perform "semantic" tests and/or operations. To this, the METAL parsers add considerably more powerful facilities.

First, there may be both a preprocessor to operate upon the input before the parser sees it, able to change it arbitrarily, and a postprocessor to operate upon the "output" after the parser is finished, able to change it arbitrarily. Second, lexical analysis is left to the user -- seemingly a burden at first thought, but in the end a blessing when one considers the possible (un)desirability of spelling correction, or differential treatment of languages other than, say, English. Lexical analysis is performed on each word as it is encountered during parsing by calling a certain user-definable function (which has a reasonable default). Third, and most importantly, the evaluation or interpretation of "semantic" routines -- indeed, the entire question of the parse structure (e.g., tree) -- is left to the user; this too may seem onerous until one realizes that each application may place different requirements upon the nature of a parse structure (syntax tree here, semantic structure there, database query somewhere else) and more generally upon the nature of the "semantic" tests or routines themselves (transformations here, LISP code there, templates somewhere else). The parser itself makes no committment concerning the nature or effect of "semantic" routines, or even to their existence. Other features, explained below, offer even more flexibility in its application to language problems. After a brief sketch of the general procedure for parsing an input, this manual will describe in more detail each function available in the basic system, and each function and variable providing special control capability to the user.

The function PARSE expects a "sentence" as input, and proceeds to interpret that input according to the currently-active grammar. First, USER_PREPROCESSOR is invoked; then the result (a LISP list) is added, word by word, to the chart. Adding a word to the chart implies calling USER_WORD with the word, and adding the result (morph by morph) to the chart; USER_WORD may produce parallel (ambiguous) interpretations, each of which is a sequence of one or more morphs to be added to the chart in order. Addition of any morph to the chart may result in a sequence matching the right-hand-side of a grammar rule; if so, the associated left-hand-side is added, provided the associated "rule body expression" (when passed to USER_RULE) results in a legal interpretation; since ANY addition may complete the right-hand-side of one or more grammar rules, the process is recursive. When the recursion dies out, the next morph is added to the chart (if there are any more), else the next word is analyzed into morphs (if there is a next word), else the parser looks for "sentential" spans in the chart and passes them to USER_POSTPROCESSOR. The value of this function is the value returned by PARSE. In the case that USER_WORD fails to produce a lexical analysis, there is another mechanism (the function USER_ADD) to handle such novelties. In the case that none of these account for a word, USER_ERROR is called, which may call for the termination of the parse effort, or perhaps an attempt to continue. This, then, is an outline of the general procedure for parsing an input.

Grammar Maintenance Facilities

These are the key words: Define, Edit, eXpunge, and Print;
Idiom(s), Rule(s), and Words;
also Value and References.

These are the key functions: DI, DR, DW;
(the names are acronyms EI, ER, EW;
based on the key words) XI, XR, XW;
PI, PR, PW;
IR, RR, WR;
also FR (Function References)
and VR (Value References).

The above functions (for those knowing INTERLISP) are NLAMBDA versions of the following LAMBDA functions:

DEFINE_IDIOM, DEFINE_RULE, DEFINE_WORDS;
EDIT_IDIOM, EDIT_RULE, EDIT_WORDS;
EXPUNGE_IDIOM, EXPUNGE_RULE, EXPUNGE_WORDS;
PRINT_IDIOMS, PRINT_RULES, PRINT_WORDS;
IDIOM_REFERENCES, RULE_REFERENCES, WORD_REFERENCES;
also FUNCTION_REFERENCES and VALUE_REFERENCES.

A Rule is composed of a left-hand-side "father" category symbol, a sequence of right-hand-side "sons" categories (or quoted words), and a rule body expression. An Idiom is like a rule, except that all the sons are understood to be words (terminals), and the rule body expression (definition) is not expected to be evaluated (though it may be; see USER_IDIOM, below). Words are entries in the lexicon; they may be associated with values (definitions). In addition to the above, PARSE notes the datatype (TYPENAME) of each item in a sentence; if its TYPENAME appears as the CAR of an element of the ADD_INTRINSICS assoc-list, the item will be interpreted as an instance of the category(s) in the CDR. Currently the parser knows these TYPENAMEs: LITATOM, STRINGP, LISTP, SMALLP, FIXP, FLOATP, and OTHER. The default value of ADD_INTRINSICS is

((SMALLP NUMBER) (FIXP NUMBER) (FLOATP NUMBER)).

Finally, if the value of ADD_UNKNOWNS is not NIL, that value is taken as a list of categories to which all unknown LITATOM terminals (i.e., those words not previously accounted for) will be temporarily assigned; if the sentence is successfully parsed, DEFINE_WORDS may be called to make the assignment permanent to all categories in ADD_UNKNOWNS (see USER_POSTPROCESSOR).

The user normally proceeds by Defining Rules, Idioms, and/or Words, and then testing out the resultant grammar -- perhaps Editing parts of it and/or eXpunging parts, as well as adding more Rules, Idioms, and/or Words -- by attempting to PARSE representative sentences. To do this, one calls various Define, Edit, and/or eXpunge functions with appropriate arguments; one might also make use of the various grammar-explication functions to see, for example, where certain symbols are used in the grammar, or to see the current rule set associated with a particular category symbol. When a reasonably stable and/or large grammar is acquired, one can direct INTERLISP to dump it onto a file, where it may be reloaded later for further use and/or development.

The syntax of the grammar maintenance function calls is as follows:

DI [cat (words...) value] -- Defines an Idiom (list of words) in the category `cat´ with the definition `value´ [cf. USER_IDIOM]. An idiom may be considered to be a multi-word lexical entry.
Ex: (DI NP (THE UNITED STATES) (COUNTRY . USA))

DR [cat (symbols...) expr] -- Defines a Rule (list of symbols) in the category `cat´ with "semantic" interpretation provided by evaluating `expr´ when PARSE finds a symbol sequence in the chart matching the list `(symbols...)´ [cf. USER_RULE]. Any/every symbol in the list which is enclosed in quotes (") is a terminal (word) rather than a category symbol.
Ex: (DR NP ("THE" ADJ N) (DEF-NP ADJ N))

DW [cat word (word . value)...] -- Defines Words (lexical entries) in the category `cat´. Each `word´ may be paired with a `value´ (default = the word itself); one may associate MORE than 1 `value´ per `word´ by using "sense numbers" as the CARs of the values [cf. USER_WORD]. The default sense no. is 0.
Ex: (DW N (MCS (ISA . SHIP) (TYPE EQ ´MCS´))
(MCS 1 (ISA . ATTR) (DBATTR . MCS)))
NOTE that the multiple senses must correspond 1-1 with words; i.e., the `word´ must be individually paired with each value it is to take as a "sense meaning", and each such pair must have a different "sense number" as the CAR of the `value´.

EI [cat idioms-to-be-edited] -- used to Edit (perhaps only certain) Idioms associated with the l-h-s "father" category `cat´. If no Idioms are specified (i.e., only the `cat´ arg is passed), all idioms associated with category `cat´ are brought up for editing.
Ex: (EI NP (THE UNITED STATES))

ER [cat rules-to-be-edited] -- used to Edit (perhaps only certain) Rules associated with the l-h-s "father" category `cat´. If no Rules are specified (i.e., only the `cat´ arg is passed), all rules associated with category `cat´ are brought up for editing.
Ex: (ER NP ("THE" ADJ N))

EW [cat words-to-be-edited] -- used to Edit (perhaps only certain) Words associated with the l-h-s "father" category `cat´. If no Words are specified (i.e., only the `cat´ arg is passed), all words associated with category `cat´ are brought up for editing.
Ex: (EW N MCS)

XI [cat (words...) value] -- used to eXpunge an Idiom (list of words). If no idiom is specified (i.e., only the `cat´ arg is passed), then all idioms in category `cat´ are deleted.
Ex: (XI NP (THE UNITED STATES) (COUNTRY . USA))

XR [cat (symbols...) expr] -- used to eXpunge a Rule (list of symbols). If no rule is specified (i.e., only the `cat´ arg is passed), then all rules in category `cat´ are deleted.
Ex: (XR NP ("THE" ADJ N) (DEF-NP ADJ N))

XW [cat words-to-be-expunged] -- used to eXpunge (lexical) Words; all senses of each word are deleted. If no words are specified (i.e., only the `cat´ arg is passed), then all words in category `cat´ are deleted.
Ex: (XW N MCS)

PI [cat flg] -- Prints all Idioms in category `cat´, and their associated values if `flg´ is not NIL.
Ex: (PI NP T)

PR [cat flg] -- Prints all Rules in category `cat´, and their associated rule body exprs if `flg´ is not NIL.
Ex: (PR NP T)

PW [cat flg] -- Prints all Words in category `cat´, and their associated values if `flg´ is not NIL.
Ex: (PW N T)

IR [word flg] -- prints all Idiom References to `word´, and their associated values if `flg´ is not NIL.
Ex: (IR THE T)

RR [sym flg] -- prints all Rule References to `sym´ ("-ed if a word), and their associated rule body exprs if `flg´ is not NIL.
Ex: (RR "THE" T)

WR [word flg] -- prints all Word (lexical) References to `word´, and their associated values if `flg´ is not NIL.
Ex: (WR MCS T)

FR [fn-name flg] -- prints all rules where the named Function is Referenced, and their associated rule body expressions (whose CARs = fn) if `flg´ is not NIL.
Ex: (FR DEF-NP T)

VR [value] -- prints all rules, idioms, and words (lexical entries) where `value´ is Referenced (returned).
Ex: (VR (COUNTRY . USA))

Note that Defining a Rule or an Idiom that is already defined with a different rule body expression or value does NOT redefine the old one! It simply adds the new definition. The parsers both produce all interpretations, and assume ambiguity is OK. Thus to replace a definition one must (1) eXpunge the old one, or (2) [better] use the appropriate Edit function. Ex:

```
(ER S (NP VP) (NP VP PP))
```

will edit the 2 rules S -> NP VP and S -> NP VP PP -- along with their rule body expressions. When using EI, ER, or EW, one may delete all BUT ONE of the definitions if desired; the LISP editor won´t allow deletion of the last one. The (potential) ambiguities w.r.t. Words is handled differently, using sense numbers as mentioned above. One CAN reDefine a sense of a Word to return a new value via DW, using the appropriate number (default 0) for the sense being redefined. The acceptance of ambiguous grammars is why the rule body expr or value must be supplied to XI and XR. These considerations make the editor more likely to be used than one might expect a priori.

Parsing

PARSE is the top-level function; its arg is a list of words (the sentence). Its value is that returned by USER_POSTPROCESSOR. It works bottom-up, left-right, by adding one word at a time to the CHART -- in terms of its morphemes´ category symbols and their definitions (interpretations) -- and then exploring the "ramifications" of the addition. If an addition results in the completion of the right-hand-side of a rule, then a new arc is constructed spanning the CHART´s corresponding "right-hand-side arcs" with the category on the rule´s left-hand-side as its phrase marker. This construction, however, depends on the USER´s "semantic" RULE function returning an interpretation other than (the value of) *ERROR* when invoked with the l-h-s FATHER symbol, the r-h-s SON symbols, and their corresponding interpretations. At the end of the sentence, PARSE looks for one or more spans in the chart from the last node to the first, whose phrase markers are in the list ROOTCATEGORIES; these spans, if any, constitute the interpretations of the sentence, and are passed to USER_POSTPROCESSOR for any processing the user desires.

The parser makes no commitment to any parse structure (such as a tree). In fact, it does not produce anything but a chart -- which contains insufficient information to compute a parse tree, or much of anything else. In this way it shifts the burden of producing the desired output structure onto the grammar writer. While this might not at first glance seem reasonable, it becomes so when considering that, by not building anything by default, the system will not burden the user with building a structure (such as a parse tree) that is unnecessary for his particular application. (For instance, some applications require only the construction of a database query from an English input, and have no use at all for syntactic structures.) Even in applications where a parse tree is desired, there may be differences of opinion as to the best parse tree structure (e.g., GENSYMs and property lists vs. ASSOC-lists vs. multi-word plexes [records] with assigned slots). This approach, then, seems most reasonable a priori. Every necessary facility is provided whereby the grammar writer may cause a structure appropriate to his application to be created; specifically, what is called a "semantic" interpretation of a phrase is constructed by code written by the user. It may for example be a parse tree. The parser in no way considers the content or structure of what that interpretation is, other than to determine whether is is EQ to *ERROR* (which implies an error condition, i.e., rejection), but associates the interpretation with the arc it builds in the chart so that it may be passed back to the interpretation routine USER_RULE at appropriate times. Finally, the interpretations associated with the ROOTCATEGORIES arcs spanning the input will be passed to USER_POSTPROCESSOR for ultimate disposition.

The user may work with any number of disjoint grammars and lexicons in a single session (e.g., in a Machine Translation system where different natural languages require different grammars). To effect the switch from one grammar to another, the function GRAMMAR is provided. The argument -- any literal atom the user desires -- "names" the grammar to be employed or defined. In addition to SETting some variables special to the parser, GRAMMAR binds the global variables *GRAMMAR* and *LEXICON* to its arg. The initial value of *GRAMMAR* is `CKY´ or `LC´, depending on the particular parser employed. In order to specify which grammatical categories are to be considered as "sentences," the function SET_ROOTCATEGORIES is provided. The argument is a list of one or more categories from the currently-selected grammar to act as ROOTCATEGORIES. ROOTCATEGORIES are phrase-markers spanning 1 or more sentences in the language defined by the grammar; each ROOTCATEGORY symbol will appear as the `father´ of at least one Rule, Idiom, or Wordlist defined by DR, DI, or DW, or appear in ADD_UNKNOWNS or ADD_INTRINSICS. The initial value of ROOTCATEGORIES is `(S)´. In addition, ROOTCATEGORY is bound to (CAR ROOTCATEGORIES): this is the "father" symbol which various system functions default to if a NIL argument is provided. SET_ROOTCATEGORIES returns as its value the previous value of ROOTCATEGORIES.

Each of the METAL bottom-up parsers is augmented by "top-down filtering." Top-down filtering is the restriction of the rules applied by a bottom-up parser to those that would be applied by a top-down parser; in other words, a bottom-up parser with top-down filtering will apply no rules that a good top-down parser would fail to apply. Good top-down parsers are noted for their restriction of the rules they apply based on the input string (sentence), but have various drawbacks such as an inability to effectively deal with left-recursive syntax rules and lack of great speed in discovering rules to apply; bottom-up parsers, while discovering applicable rules very quickly, have the drawback of applying many rules needlessly (i.e., applying rules which could not possibly contribute to a sentence-level analysis). Top-down filtering, then, implies a combination of both strategies in the hope that the speed of rule application associated with the bottom-up strategy will be abetted by the restriction on rules supplied by the top-down strategy. The advantages of filtering have been demonstrated to depend on sentence length, and may depend on other factors as well. The use of filtering is strictly controlled by (the value of) the global variable TOP-DOWN_FILTER (see below).

User-definable Functions

USER_ADD [FATHER SON] is called by the parser when it encounters in the sentence a numeric literal matching a constant in some grammar rule or idiom, or an item whose datatype TYPENAME is in ADD_INTRINSICS, or a literal atom not accounted for by any other mechanism. FATHER is the category (from ADD_INTRINSICS or ADD_UNKNOWNS) or literal to be added to the CHART. SON is the item in the sentence. By returning (the value of) *ERROR* USER_ADD signals the proposed syntactic interpretation to be unacceptable; it will thus not be added to the CHART. Otherwise, the value returned by USER ADD is taken as the `semantic´ interpretation of the proposed -- and accepted -- syntactic interpretation wherein FATHER dominates SON.
Default: return SON.

USER_ERROR [PARSED WORD REST] is called when an unknown WORD is encountered. PARSED is the (possibly empty) list of words understood so far; WORD is the word which cannot be accepted; REST is the (possibly empty) list of words remaining in the sentence. USER ERROR is user-definable, and may do anything the user likes -- including defining WORD somehow, or substituting another word. If USER_ERROR returns (the value of) *ERROR* then parsing will terminate and USER_POSTPROCESSOR will be called immediately. Otherwise (NOTE!) the value is taken as the word to try in place of WORD (though it may be WORD if WORD is defined herein) and the parser will continue. NOTE that, if WORD is returned without some necessary corrective action (e.g., defining WORD) an INFINITE LOOP may result -- i.e., WORD will still be unknown, and the parser will call USER_ERROR again with the same arguments.

```
Default: (PRIN1 *GRAMMAR*)
         (COND [PARSED (PRIN1 " does not permit ")
                       (PRIN1 WORD)
                       (PRIN1 " to follow ")
                       (MAPRINT PARSED)
                       (TERPRI)]
               [(PRIN1 " does not accept the 1st word: ")
                (PRINT1 WORD)])
        then return *ERROR*.
```

USER_IDIOM [FATHER SONS BINDS VALUE] is called when the parser finds an IDIOM in the sentence. FATHER is the l-h-s symbol in the idiom´s definition; SONS are the r-h-s symbols (words) in the definition; BINDS are the "semantic" interpretations of the respective SONS [cf. USER_WORD]; VALUE is the "semantic" interpretation from the idiom´s definition [cf. DI]. It is expected that, for most applications, USER_IDIOM will simply return VALUE. However, the user may perform any operation he desires, such as reformatting VALUE, or calling some other function. The value returned by USER_IDIOM will be treated as the "semantic" interpretation of the idiom in the usual way;

i.e., the parser will ignore it other than associating it with the FATHER symbol in the CHART and (perhaps) passing it as an element of BINDS when calling USER_RULE later.
Default: return VALUE.

USER_NEW WORD [W] is called with every word, rule literal and idiom constituent, the first time such is encountered in the definition of a grammar. In the latter two cases, the arg W will be a string, number, etc, but not a literal atom; in the case of words, it will be a literal atom as required by DEFINE_WORDS. USER_NEW_WORD may do anything the user likes, including nothing. It may operate to help his functions USER_WORD and/or USER_PREPROCESSOR when they look for lexical analyses of the word by, for instance, constructing a "letter-tree" to support spelling correction. Default: just return W.

USER_POSTPROCESSOR [INTERPS INFO] is called with two args: a list of "semantic" INTERPretationS, if any, as returned by the ROOTCATEGORIES' rule body expression(s); and either the (perhaps partial) CHART built by PARSE if that process was unsuccessful (not normally of much help to users), or the (perhaps empty) list of UNKNOWN words encountered during a successful parse, with their "semantic" interpretations as returned by USER_ADD in a format suitable for passing as the second argument to DEFINE_WORDS. If the INTERPS list is empty, then PARSE failed to find any interpretations with a spanning arc from the list ROOTCATEGORIES. (The linguist might wish to view the INFO chart in this case.) The user might, e.g., wish to use USER_POSTPROCESSOR to print out and/or disambiguate interpretations, print the processing time, bind the inputs to some global variables, and/or Define the previously unknown Words in INFO using the category(s) in ADD_UNKNOWNS.

```
Default: (SETQ PARSETIME (DIFFERENCE (CLOCK 2) PARSETIME))
         (AND [SETQ INTERPRETATIONLIST INTERPS]
              INFO
              [MAPC ADD_UNKNOWNS
                    (FUNCTION (LAMBDA (CAT)
                                      (DEFINE_WORDS CAT INFO)))])
         (SELECTQ [SETQ INFO (LENGTH INTERPRETATIONLIST)]
                  [0 (PRIN1 "0 interpretations; ")
                     (PRIN1 PARSETIME)
                     (PRIN1 " milliseconds.")]
                  [1 (PRIN1 "1 interpretation in ")
                     (PRIN1 PARSETIME)
                     (PRIN1 " milliseconds.")]
                  [PROGN (PRIN1 INFO)
                         (PRIN1 " interpretations in ")
                         (PRIN1 PARSETIME)
                         (PRIN1 " milliseconds:  ")
```

```
                (PRIN1 (QUOTIENT PARSETIME INFO))
                (PRIN1 " msecs/interp.")])
        (TERPRI)
        then return INTERPS.
```

USER_PREPROCESSOR [L] has one crack at each input to PARSE before PARSE sees it. The output must be a list -- presumably of words, though PARSE could care less. Then, before each word is added to the CHART by PARSE, USER_WORD is called with that word. This function might be used, for example, to set TOP-DOWN_FILTER to T or NIL depending on sentence length.

```
Default: (SETQ PARSETIME (CLOCK 2))
        then return L.
```

USER_RULe [FATHER SONS BINDS EXPR] is called when the parser matches a RULE from the CHART. FATHER is the l-h-s in the rule´s definition; SONS is a list of the r-h-s symbols in the rule´s definition; BINDS is a list of the "semantic" interpretations of the respective SONS; EXPR is the expression associated with the rule. By returning the value of *ERROR*, USER_RULE signals the proposed syntactic interpretation to be unacceptable; it will therefore not be added to the CHART. Otherwise, the value returned by USER_RULE is taken as the "semantic" interpretation of the proposed — and accepted — syntactic interpretation.

```
Default: (MAP2C SONS BINDS
                (FUNCTION (LAMBDA (USER_S USER_B)
                                  (AND [LITATOM USER_S]
                                       [SET USER_S USER_B]))))
        then return (EVAL EXPR).
```

USER_WORD [W REST] is responsible for Word analysis, including `literals´. (Literals are strings in idioms and grammar rules.) W is the word [actually sentence item, whether word (LITATOM), number, quotation (STRING), or parenthetical (LIST) expression]; REST is the remainder of the sentence (after W) -- useful, e.g., when deciding whether to separate-out punctuation. The output of USER_WORD, if not NIL, is interpreted as a list of ALTERNATIVE analyses which MUST have one of the following forms (where the asterisk `*´ indicates the format created by the DEFINE functions):

```
     (analysis1 analysis2 ... analysis-n)* or (cat . senses)
```

The analyses are considered to be parallel alternatives, each of which MUST have one of the following forms:

```
     (morph1 ... morph-n) or (cat . senses)*
```

The morphs will be CHARTed in the serial order specified. NOTE: the morph-i lists may be destructively altered by the parser; therefore, they must be newly CONStructed for each analysis. Fach morph must have one of the following forms:

```
     (interp1 ... interp-n) or (cat . senses)
```

The interps are considered to be parallel alternatives, each of which must have the form:

    (cat . senses)

If `cat´ = (the value of) LITERAL_CATEGORY, then `senses´ corresponds to a literal in the grammar (created as a side-effect of DEFINE_IDIOM or DEFINE_RULE); it will be treated like a category symbol and CHARTed using W as its `value´ (see below). Otherwise, `cat´ is a grammatical category and `senses´ must have the form:

    (sense1 ... sense-n)

The senses are considered to be parallel alternatives, each of which must have the form:

    (sns# . value)

The values are added to the CHART (in parallel) under the associated category, as would happen if USER_RULE returned the value in response to a call wherein FATHER was bound to the associated category. The default below, for example, returns LITERAL and WORD analyses for the word W in the format

    ((cat . senses) ... (cat . senses))

else NIL if there are none.

Default: return (GETP W *LEXICON*).

Global Variables

ERROR is the "error" flag; if/when USER_ADD, USER_ERROR, USER_IDIOM, and/or USER_RULE return (the value of) this variable, special action is taken (see those 4 functions for details). The default value of *ERROR* is itself; the user may change it as desired -- for example, to NIL to disallow NIL as a "semantic" interpretation of a phrase.

GRAMMAR and *LEXICON* are both bound to the name of the currently active grammar. (See the function GRAMMAR for details.)
Default: `CKY´ or `LC´, depending on the particular parser.

ADD_INTRINSICS is bound to an assoc-list, the CARs of whose elements are names of LISP´s intrinsic datatypes -- currently chosen from among LITATOM, STRINGP, LISTP, SMALLP (for "small" integers up to 1000 or so, depending on LISP implementation details), FIXP (larger integers), FLOATP, and OTHER (for anything else -- not very likely to be encountered). The CDRs of those elements are to be lists of grammatical categories to which input items of the matching datatype will be assigned -- regardless of whether the item was found in the lexicon. (See the discussion of ADD_INTRINSICS above, plus the function USER_ADD, for more details.)
Default: ((SMALLP NUMBER) (FIXP NUMBER) (FLOATP NUMBER)).

ADD_UNKNOWNS is bound to a list of grammatical categories to which an unknown word (not handled lexically or through ADD_INTRINSICS) is to be assigned. The default is (PROPN). If set to NIL, for example, unknown words will result in a call to USER_ERROR when encountered. If ADD_UNKNOWNS is not NIL -- and it must then be a list -- then USER_ERROR may never be called. (See ADD_UNKNOWNS and USER_ADD for more details.)

CKY_FUNCTIONS, CKY_GRAMMAR, CKY_GRAMMARCOMS, CKY_IDIOMS, CKY_RULES, CKY_TREE, and CKY_WORDS are reserved symbols -- as are ALL symbols starting with "CKY_" or "cky_".

FUN_SIZE is bound to a number that determines whether a rule body expr is to be made into a function. For "production" runs, a system will generally benefit from compilation of the larger rule body exprs, but to be compiled they must become functions. On the other hand, compiling very small exprs may be wasteful. When a grammar rule is defined, the size (number of list cells) of its rule body expression is compared with FUN_SIZE; if FUN_SIZE is larger, the expression remains unchanged; otherwise the expression becomes a call to a generated function whose definition is that rule body expression. Thereafter, when the grammar is dumped using the MAKEFILE cum CKY_GRAMMARCOMS [or LC_GRAMMARCOMS] facility, the resultant file may be compiled to produce a (.COM) file containing the grammar and lexicon plus compiled functions, which may then be loaded in place of the original grammar source file. Default value: 10.

GENNUM is a pseudo-reserved symbol; certain LISPs (like INTERLISP) use this as a GENSYM initializer, and the parser treats it in a similar fashion. The user should not use GENNUM.

GRAMMAR_SOURCE is the flag which controls whether the "source" form of a grammar (cum lexicon) is to be maintained. This maintenance is necessary, for example, if the user wishes to edit or expunge rules, idioms, or words. For "production" runs it may be desirable to set GRAMMAR_SOURCE to NIL before loading a grammar, to save space. The default value is T, which means that source forms are saved and changes may be effected in the grammar at run time. GRAMMAR_SOURCE should be changed, if at all, BEFORE loading the grammar.

LC_FUNCTIONS, LC_GRAMMAR, LC_GRAMMARCOMS, LC_IDIOMS, LC_RULES, LC_STARTEES, LC_STARTERS and LC_WORDS are reserved symbols -- as are ALL symbols beginning with "LC_" or "lc_".

LITERAL_CATEGORY is bound to the name of the grammatical pseudo-category into which literals (in grammar rules) and idiom constituents are inserted so that USER_WORD may find them. The value of LITERAL_CATEGORY (default = LITERAL) must NOT correspond to any true lexical category, since -- like lexical categories -- it will appear as the CAR of an element on the word´s lexical-entries assoc-list; unlike lexical categories, however, (the value of) LITERAL_CATEGORY is not CHARTed as a category in the lexical analysis, but rather its associated literal (the CDR of the pair) is, using the current word as its value (definition). [See the description of USER_WORD for more details]. The user may, if desired, reset LITERAL_CATEGORY to any symbol, but must do so before any grammars are loaded and not afterwards.

ROOTCATEGORY is bound to (CAR ROOTCATEGORIES) by the function SET_ROOTCATEGORIES. This is the category to which the various EDITing and PRINTing functions default when no argument is provided.

ROOTCATEGORIES is bound to the list of grammatical/lexical categories which are acceptable (to the user) in accounting for an input [else to `T´ indicating that all categories are acceptable]. In other words, "root" categories define what a "sentence" is in terms of the currently active grammar (see the function SET_ROOTCATEGORIES for more details). Default value: (S).

TOP-DOWN_FILTER is a flag which controls the activity known as "top-down filtering." If filtering is desired, this variable must be set non-NIL BEFORE (!) the relevant grammar is defined, in addition to its being non-NIL when parsing. If NIL, no filtering is performed. Once a grammar has been defined with TOP-DOWN_FILTER non-NIL, filtering can be turned on or off as desired by toggling TOP-DOWN_FILTER. Default value: NIL.

Appendix I

Commands to Create, Test and Save a Sample Grammar
for sentences like: THE OLD MAN ATE FISH

```
@LISP

_(LOAD ´<LRC.MT>CKY.COM)                        (* or <LRC.MT>LCP.COM)

_(GRAMMAR ´ENGLISH)                             (* name the grammar)

_(DR S (NP VP) (LIST ´S NP VP))                         (* define S rule)

_(DR NP (DET NP) (LIST ´NP DET NP))             (* define NP rules)

_(DR NP (ADJ NP) (LIST ´NP ADJ NP))

_(DR NP (N) (LIST ´NP N))

_(DR VP (V NP) (LIST ´VP V NP))                         (* define VP rule)

_(DI NP (THE MAN) (NP THE MAN))                         (* define NP idiom)

_(DW DET (A DET . IND) (THE DET . DEF))                 (* define DETerminers)

_(DW ADJ OLD)                                           (* define an ADJective)

_(DW N (FISH N . FISH) (MAN N . MAN))                   (* define Nouns)

_(DW V (ATE V EAT -ED))                                 (* define a Verb)

_(SET_ROOTCATEGORIES ´(S))                      (* choose ROOTCATEGORY)

_(PARSE ´(THE MAN ATE A FISH))                          (* test the grammar)

_(SETQ TEST-GRAMCOMS CKY_GRAMMARCOMS)                   (* or LC_GRAMMARCOMS)

_(MAKEFILE TEST-GRAM.LSP)                       (* save the grammar)
```

Appendix II

The Sample Grammar File (suitable for LOADing)
resulting from the commands in Appendix I

```
(FILECREATED "11-Feb-81 16:44:43" <LRC.PUBLIC>TEST-GRAM.LSP.1 1865    )


(PRETTYCOMPRINT TEST-GRAMCOMS)

(RPAQQ TEST-GRAMCOMS [(FNS * (SORT (GETP (QUOTE CKY_FUNCTIONS)
                                        *GRAMMAR*)))
     (P
       *
       (LIST
         (LIST (QUOTE GRAMMAR)
               (KWOTE *GRAMMAR*))
         (LIST (QUOTE PUT)
               (QUOTE (QUOTE CKY_FUNCTIONS))
               (KWOTE *GRAMMAR*)
               (KWOTE (GETP (QUOTE CKY_FUNCTIONS)
                            *GRAMMAR*)))
         [CONS (QUOTE DRQ)
               (MAPCAR (SORT (GETP (QUOTE CKY_RULES)
                                   *GRAMMAR*))
                       (FUNCTION (LAMBDA (R)
                                   (CONS R (GETP R CKY_RULES]
         [CONS (QUOTE DIQ)
               (MAPCAR (SORT (GETP (QUOTE CKY_IDIOMS)
                                   *GRAMMAR*))
                       (FUNCTION (LAMBDA (I)
                                   (CONS I (GETP I CKY_IDIOMS]
         [CONS
           (QUOTE DWQ)
           (MAPCAR
             (SORT (GETP (QUOTE CKY_WORDS)
                         *GRAMMAR*))
             (FUNCTION
               (LAMBDA
                 (S)
                 (CONS S
                       (MAPCONC
                         (SORT (GETP S CKY_WORDS))
                         (FUNCTION
                           (LAMBDA (W)
                                   (MAPCAR (CDR (FASSOC S
                                                        (GETP W
                                                       *LEXICON*)))
                                           (FUNCTION
                                             (LAMBDA
                                               (X)
```

```
                                                    (COND
                                                      ((NEQ 0 (CAR X))
                                                       (CONS W X))
                                                      ((EQ W (CDR X))
                                                       W)
                                                      ((CONS W
                                                             (CDR X]
         (LIST (QUOTE SET_ROOTCATEGORIES)
               (KWOTE ROOTCATEGORIES])
(DEFINEQ
)
(GRAMMAR (QUOTE ENGLISH))
(PUT (QUOTE CKY_FUNCTIONS)
     (QUOTE ENGLISH)
     NIL)
[DRQ (NP ((N)
       (LIST (QUOTE NP)
             N))
      ((ADJ NP)
       (LIST ´NP ADJ NP))
      ((DET NP)
       (LIST (QUOTE NP)
             DET NP)))
     (S ((NP VP)
      (LIST (QUOTE S)
            NP VP)))
     (VP ((V NP)
       (LIST (QUOTE VP)
             V NP]
[DIQ (NP ((THE MAN)
       (NP THE MAN]
(DWQ (ADJ OLD)
     (DET (A DET . IND)
       (THE DET . DEF))
     (N (FISH N . FISH)
     (MAN N . MAN))
     (V (ATE V EAT -ED)))
(SET_ROOTCATEGORIES (QUOTE (S)))
(DECLARE: DONTCOPY
  (FILEMAP (NIL (1310 1321))))
STOP
```