MASSACHUSETTS INSTITUTE OF TECHNOLOGY
A. I. LABORATORY

Artificial Intelligence
Memo No. 282

March 1973

GRAMMAR FOR THE PEOPLE: FLOWCHARTS OF SHRDLU'S GRAMMAR

Andee Rubin*

ABSTRACT

The grammar which SHRDLU uses to parse sentences is outlined in a series of flowcharts which attempt to modularize and illuminate its structure. In addition, a short discussion of systemic grammar is included.

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0003.

Reproduction of this document, in whole or in part is permitted for any purpose of the United States Government.

*NSF fellow

The purpose of these flowcharts is to make available to the general, non-SHRDLU-hacking public, the parser which SHRDLU uses. There have been many who have tried to decipher its code; most have either become hopelessly entangled or painfully made their way through its web. Now, at last, even you can have in your home a complete set of SHRDLU flowcharts. Besides making the code generally more comprehensible, the flowcharts de-emphasize the linearity of the parsing program and instead organize it into modules. Thus the reader can see the outline of a large part of the process at a fairly high level and only later turn to another page to ponder the details. Hopefully, their availability will encourage other system implementers to use the parser as a front end and will spark some cross-cultural communication between the non-computing linguistic community and AI language workers.

These flowcharts are best read with a copy of Winograd's thesis or book by your side. Although a few characteristics of systemic grammar are mentioned below, its basic philosophy is not detailed here. Neither is the design and operation of PROGRAMMAR, the language in which the parser is written. Working through the PROGRAMMAR examples in the thesis(page 153 in the thesis version or page 98 in the book) is a good preliminary exercise in following the operation of the grammar. Of course, the charts are also meant to be a guide for reading the code; they explicate the crucial sections, avoiding some of the kludges which make the grammar difficult to decipher. You should not expect, however, that the correspondence between the flow charts and either other source will be perfect. The thesis explicates a more complete grammatical

theory than that which the program embodies; in addition, feature names sometimes differ radically. While the charts stick more closely to the program than to the thesis, several sections of code have been omitted and/or "misrepresented," most notably the backtracking sections (precisely because they are mostly ad hoc). Don't be thrown by such differences. You are not missing anything or being stupid; they are (hopefully) intentional.

A FEW WORDS ABOUT SYSTEMIC GRAMMAR

Form vs. Function

A basic dichotomy which systemic grammar considers is that between the form and function of a particular unit. Form refers to the internal structure of a unit, what constituents it can have, how they must be arranged and what dependencies exist among them. Function considers a unit's place in the "higher order of things" : what roles it can play in higher constituents and what dependencies it enters into. For example, the form of an NG may be represented as several non-independent slots which need to be filled; the NG program is primarily concerned with filling those slots and amassing features (e.g. number and person). Its function is indicated by the places in other programs from which it is called: as the object of a preposition, first or second object of a verb, subject of a clause, constituent of an ADJG(e.g. as big as the ball) etc.

The form of a unit is evident in the flowcharts detailing its program; a unit's functions are indicated by the places where an attempt

to parse it is made (i.e. where

is found) There are clearly a multitude of functions for a unit. Remember that there are also many and varied forms. For example, an NG containing "anything" has a different form from the garden-variety filled-slot NG and the form of a VG cannot be easily detailed in a slot-and-filler account, though a program does a good job of describing its form.

Of course, a constituent's function may affect its form. A primary use of features in this parser is to communicate such effects up and down the tree. CLAUSE may call NG with the initial feature OBJ, which specifies objective case for any pronoun the NG may come up with. NG, in turn, may call CLAUSE with the initial features RSNG (rank-shifted noun group) ING (as in "Dancing Bulgarian dances is great fun."), which allows the clause to have no subject or a possessive subject and requires it to have an ING verb group. A good thing to focus on in reading the flowcharts is the effect of such features. Often a program will, as its first action, dispatch to pieces of code which know how such functional features affect form.

More on Features

While we're on the subject, more about the features a systemic grammar uses. So far we've been referring to their syntactic relevance and how they direct the parsing process. One of systemic grammar's claims to fame is that features have both syntactic and semantic significance, thus partially bridging the gap between syntax and semantics which has been recently so lamented. For example, the feature

DEF on an NG not only constrains its form but also tells us that a specific object or objects are being referred to (e.g. the happy smiling rabbit) The feature INT, given to clauses with intensive verbs such as BE, tells us structurally to expect an NG, ADJG or PREPG acting as a complement after the verb but also indicates that the complement will be a property of the subject of the sentence as in "The rabbit is cute." Similarly, an RSQ (rank-shifted qualifier) clause has a definite structure in which one of its constituents will be the noun it is modifying. It also gives us the semantic information that the object being referred to by the entire NG must satisfy the semantic constraints of the RSQ. Finally, a TIME NG has a very definite structure and must contain a time word such as "yesterday" or "month" (as in "last month"); it has obvious semantic significance in that we know it affects the time referred to by the utterance, which would otherwise be determined solely by the tense.

The flowcharts indicate when some of the more important features are added to nodes. (Notice that one way features are added is by sending initial features along in the attempt to parse) There are, in addition, decision points based on testing for a particular feature - the most relevant of these are also noted in the flowcharts. Most feature names are either found in the thesis or are interpretable from their spellings with a little thought. TOOB2, for example, is the feature of a verb which takes an RSNG TO as its second object as "I asked him to scratch my back." In this sentence, "him", an NG, is the first object, while the CLAUSE RSNG TO "to scratch my back" is the second. TRANSINT refers to a

verb which takes an object and a complement as "I consider him an idiot." A most important feature, REL-NOT-FOUND, is marked on an RSQ clause which has not yet determined which of its constituents' places its RELHEAD (the noun it is modifying) will take.

This treatment of features implies some uniformity in their use; actually features serve three distinct purposes in this parser. The form of a given constituent is clearly indicated by its features; its function is also so marked, although this information is somewhat redundant with the contents of the registers.(see below) All of the features relating to function could be easily derived using the registers and some simple tree-traversing. A third, very different use of features is as temporary variables during the parsing process. REL-NOT-FOUND, for example, is only temporarily attached to a clause and is not a feature referring to either function or form which remains on a node at the end of the parsing process. Such features serve to direct the parsing process but are not an end result of it.

Registers

Certain of the subparts of a constituent will always be crucial to its semantic analysis; such subparts are kept in registers so they will be readily available to the semantic specialists as well as later syntactic processes. The name of a register is in general an obvious give-away for its meaning: SUBJECT,OBJ1, LOGICAL-SUBJECT,(in passive sentences), MVB(main verb). HEAD is the register used to hold the head noun of an NG or the head adjective of an ADJG; RELHEAD holds the NG

which an RSQ modifies and is used by the RSQ CLAUSE program when it looks around for its missing constituent. Such registers add no theoretical power to the parser, since a constituent can always be found by moving around the parse tree, but they add to the ease of program writing, reading and documentation; they are another way which shows up clearly in these flowcharts for constituent programs to communicate information among themselves.

OPERATION OF THE PARSER

As explained in more detail in the thesis, the parser works, as do most such parsers, by building a parse tree. Each node is marked as to features, daughters, parent, words it contains and relevant registers. Semantics are initially present only at the word level; such definitions are stored in the dictionary. At crucial points in the grammar programs (which are indicated on the flowcharts as "CHECK SEMANTICS") semantic specialists are called.They combine the semantics of the components into a semantic structure for that part of the unit parsed so far. At least one semantic specialist exists for each syntactic unit. NG, for example, boasts two: one which handles that part of the NG through the noun, and the second which worries about the semantics of qualifiers occurring after the noun. Such a specialist can also report that it's not happy with the combination of words the parser came up with for this constituent; its complaint usually causes the unit to pop off its last-parsed constituent or, in more radical cases, to throw up its hands and

return a failure message to the program which called it. This semantic guidance prevents an attempt to parse "The girl gave the boy plants to water" analagously to "The girl gave the house plants to charity." and is one of the more interesting features of the parsing strategy.

LEGEND FOR READING THE FLOWCHARTS

Each page of the flowcharts is titled; in addition, pages are labelled with <name of unit> / <page within that unit> and such designations are used on other pages to refer to this section of the flowcharts, e.g. NG/1 or CL/10.

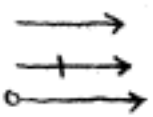
PARSE unit with given features (equivalent to a call to PARSE)

PARSE this particular word (equivalent to a call to PARSE with NIL as the first argument)

LABEL in grammar programs, local to the page it appears on.

LABEL in grammar programs. The code at this label is accessed from some other page in the flowcharts.

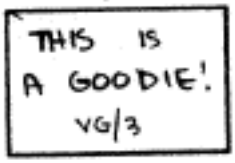
LABEL in grammar programs. The code at this label is detailed on the page indicated under the label. (Apologies for the confusing closeness of this notation and the one above it.)



Three branches leading from an attempt to PARSE
: taken if attempt succeeds
: taken if attempt fails
: taken if parse succeeds and there are no more words

left in the sentence

Branch point, relevant test is detailed inside diamond.



A large task which is detailed on another page headed by this name and designated by unit/ number below. These may be considered the primary modules of the grammar.

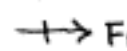
A small task, not detailed anywhere else because what it says here should be enough for you to get the idea. More details may be found in the actual code if you're interested.

FEATURE

Features being checked for are in capital letters and underlined. Unless otherwise noted, features referred to belong to the constituent currently being parsed.

an example of some text

A relevant piece of an utterance which, to be parsed, passes through this particular point in the program. The underlined constituent is the part which is particularly relevant, often what would be parsed by the nearest call to PARSE.



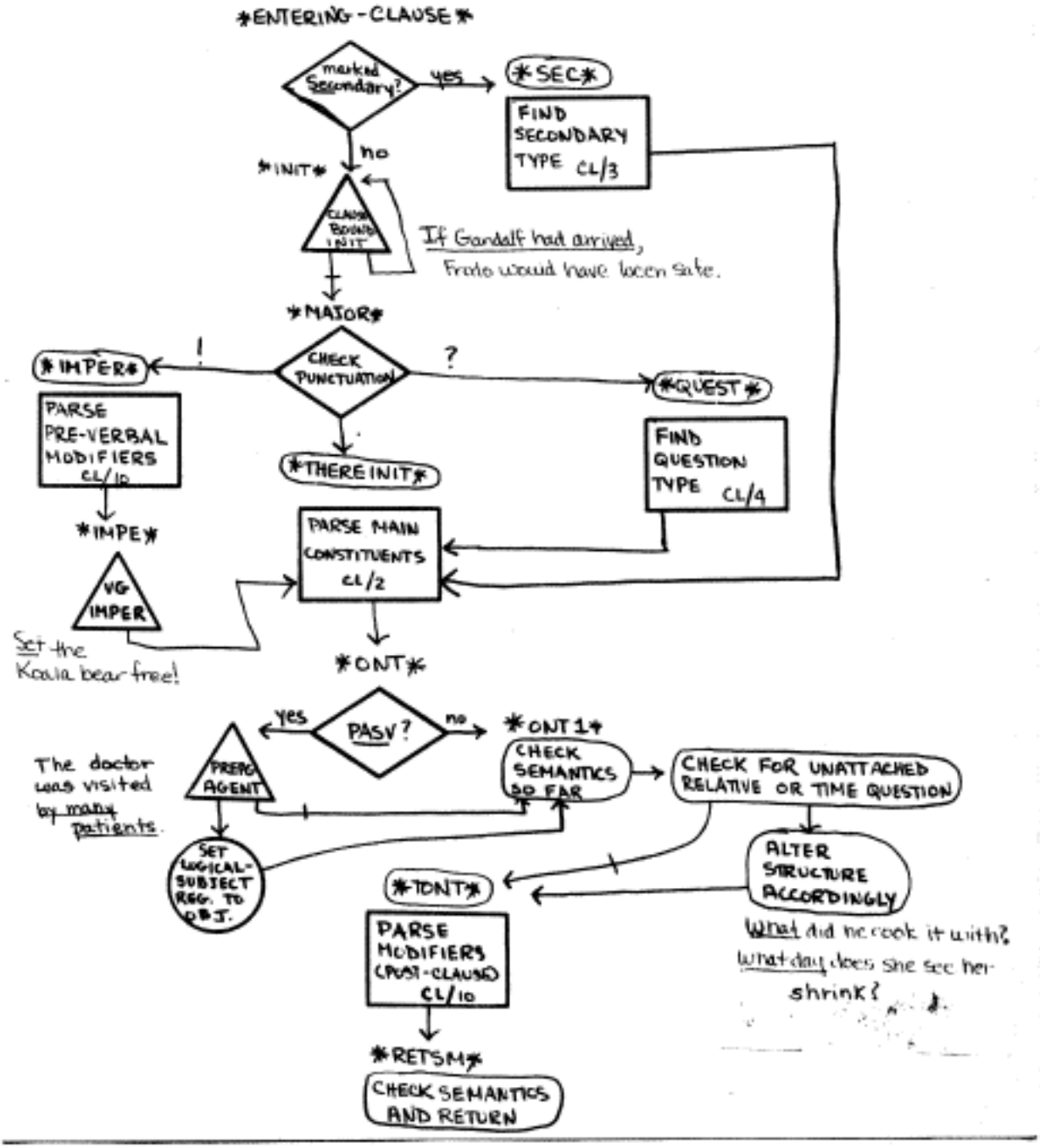
The program called to parse this constituent returns a failure message.

is the usual label for the place in each unit's program where final semantics work is done and the program returns its parsed constituent which is then attached to the tree.

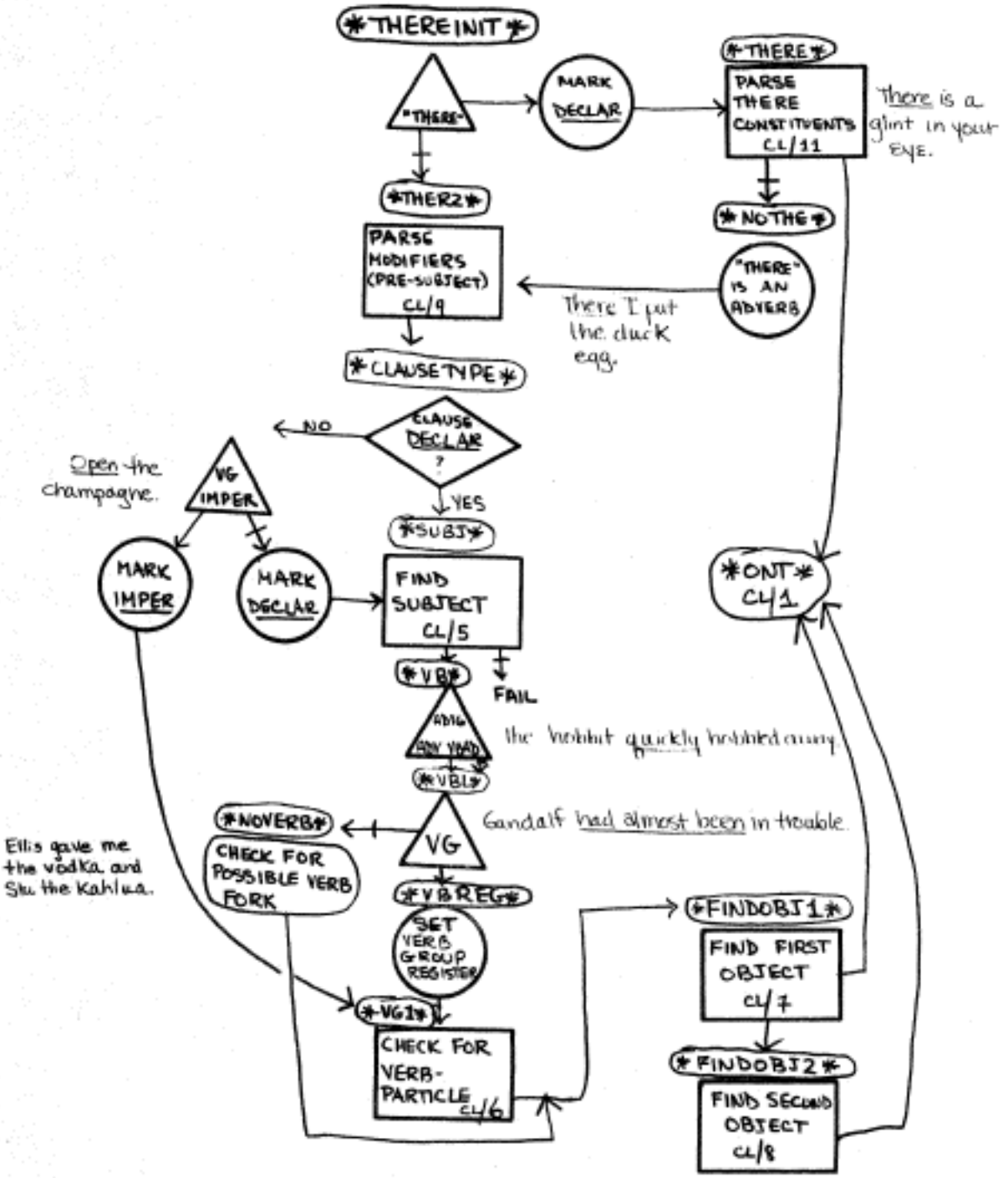
In general, the flowcharts parallel the code well enough so that the correspondences are obvious. In one case, this not true. In the flowcharts for finding the first and second object in a clause, a great deal of the work is checking features of the verb and, should the parse of a suitable object succeed, marking the clause suitably and setting an

appropriate register. This work is "hidden" in the subroutine CANPARSE, whose arguments are: number of object (1 or 2); unit to parse if the verb is appropriately marked; feature with which to mark clause if parse succeeds.

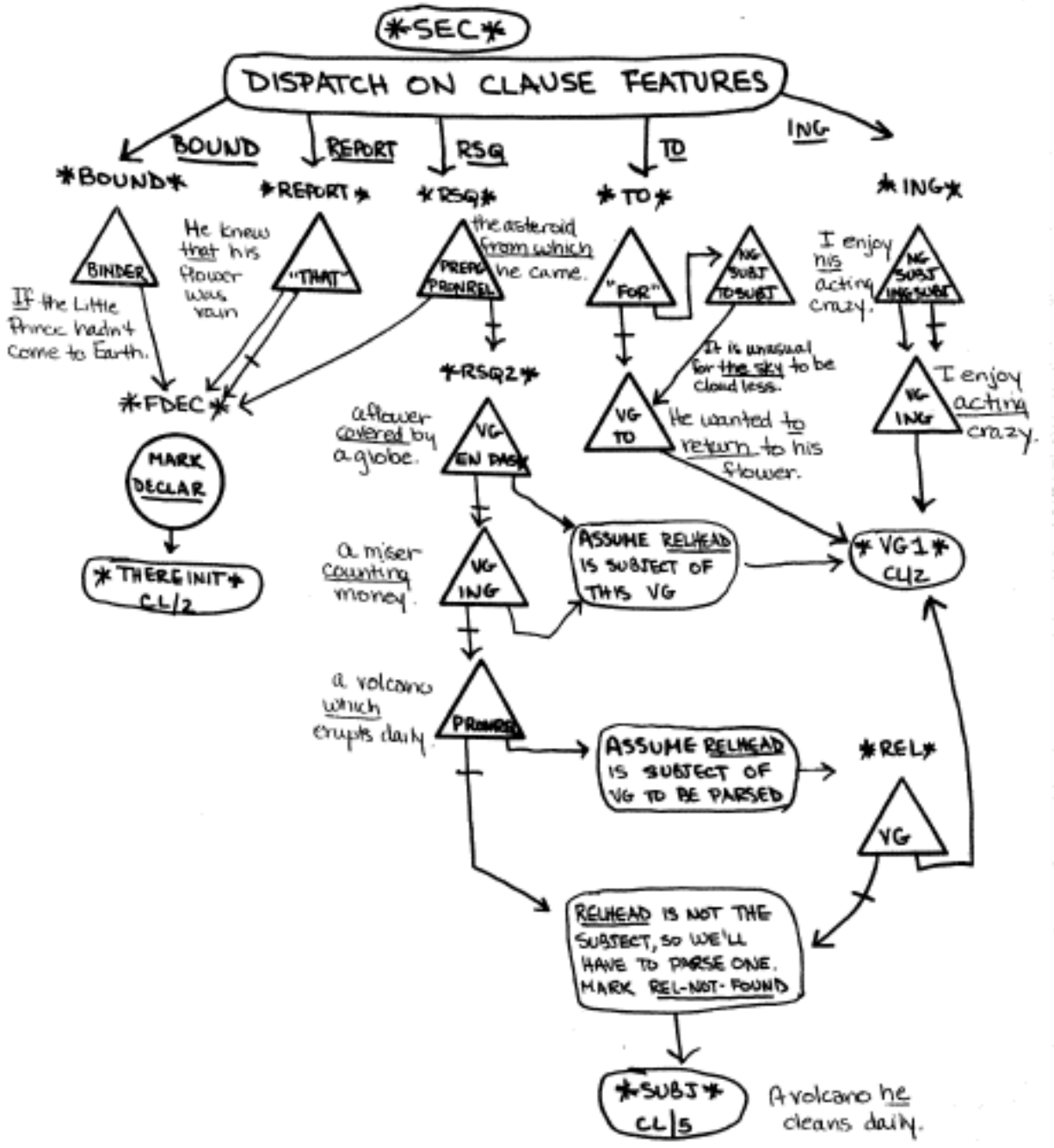
FIND CLAUSE TYPE (THIS IS THE BEGINNING - GOOD LUCK!)



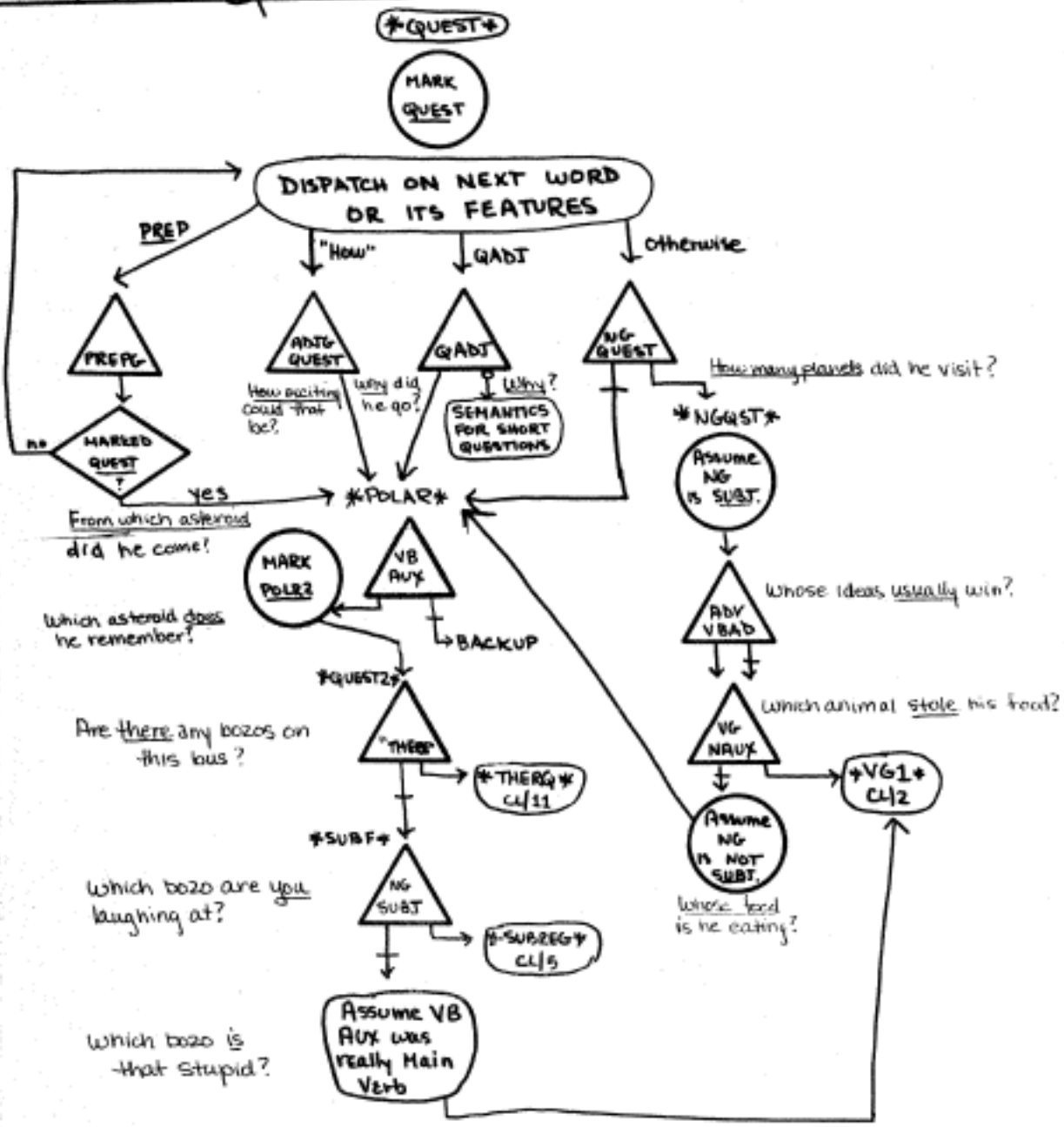
PARSE MAIN CONSTITUENTS



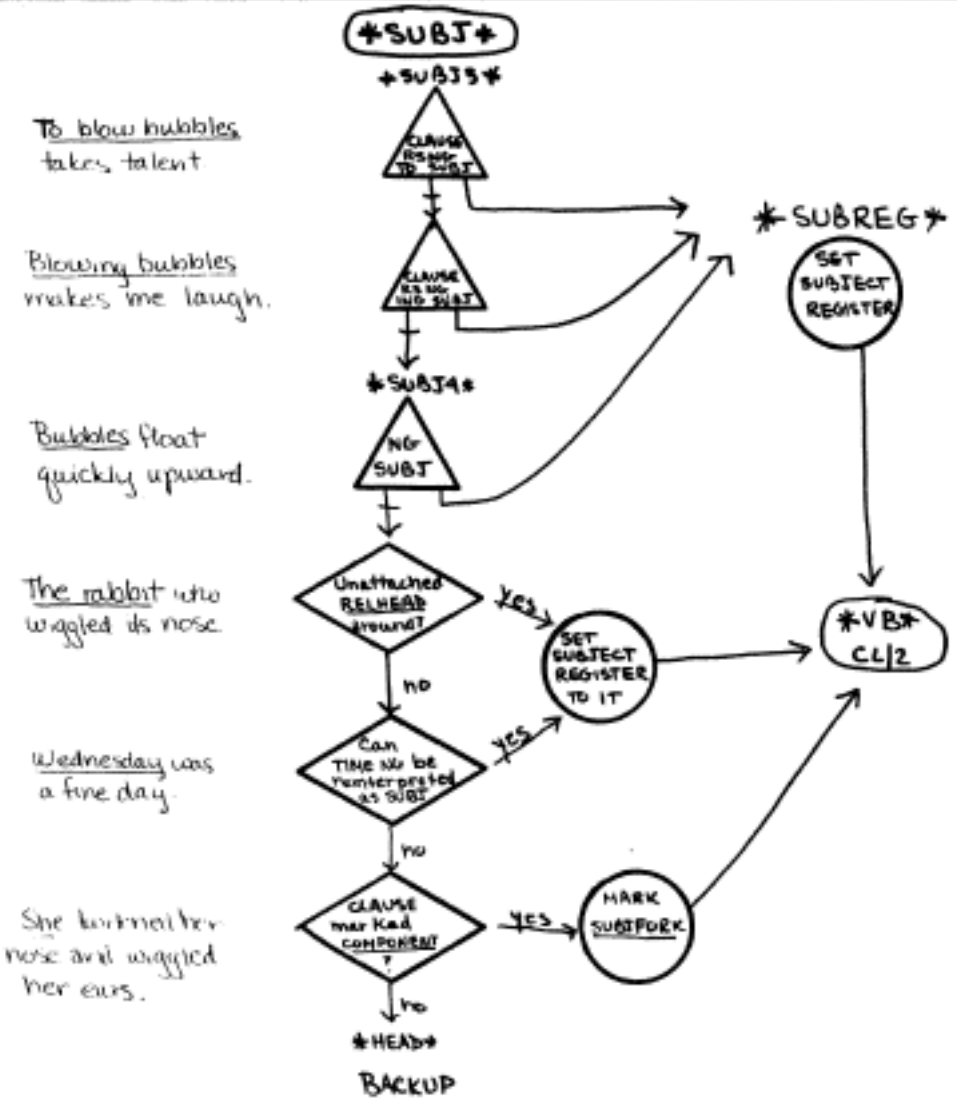
FIND SECONDARY TYPE



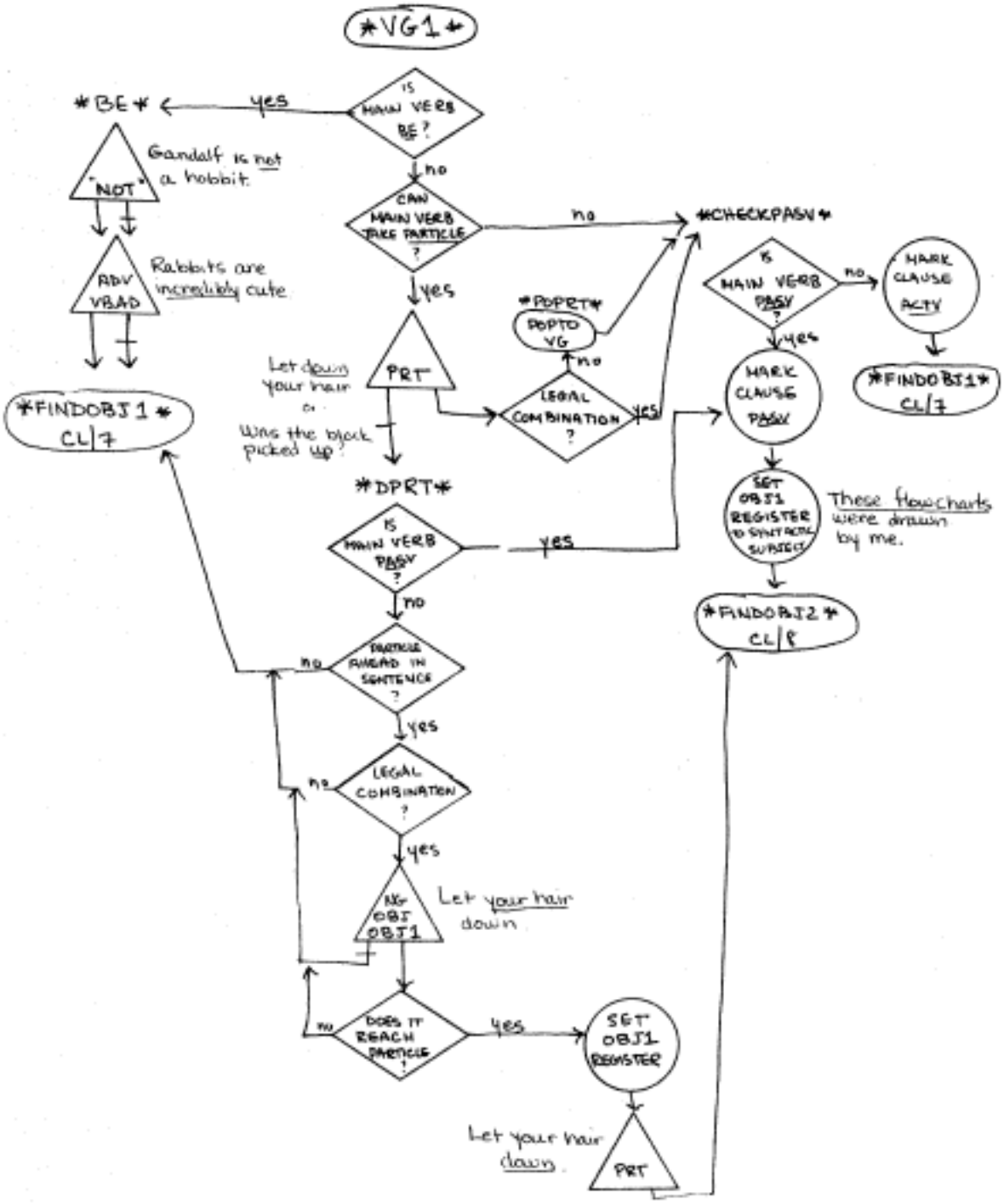
FIND QUESTION TYPE



FIND SUBJECT

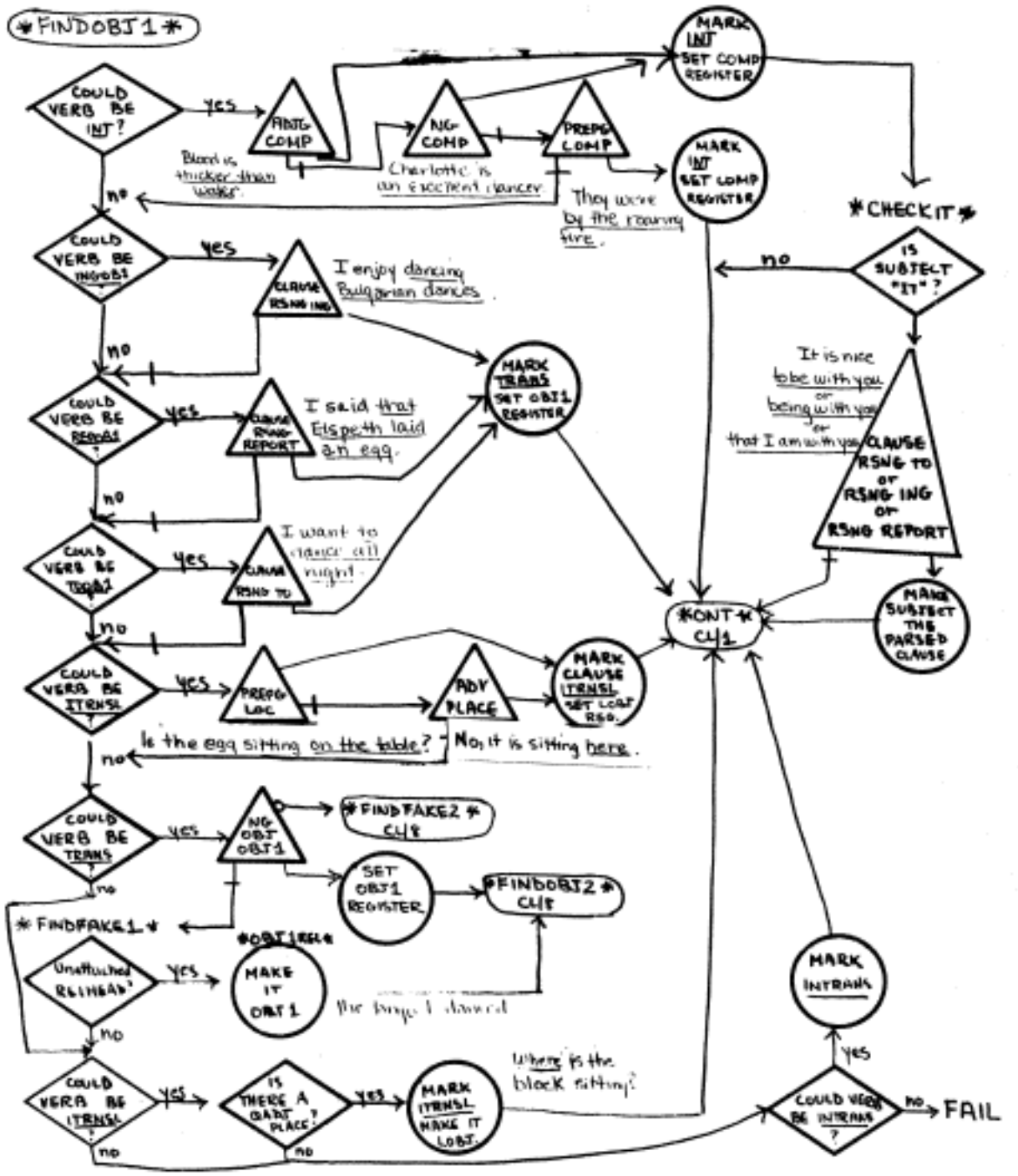


CHECK VERB-PARTICLE COMBINATION

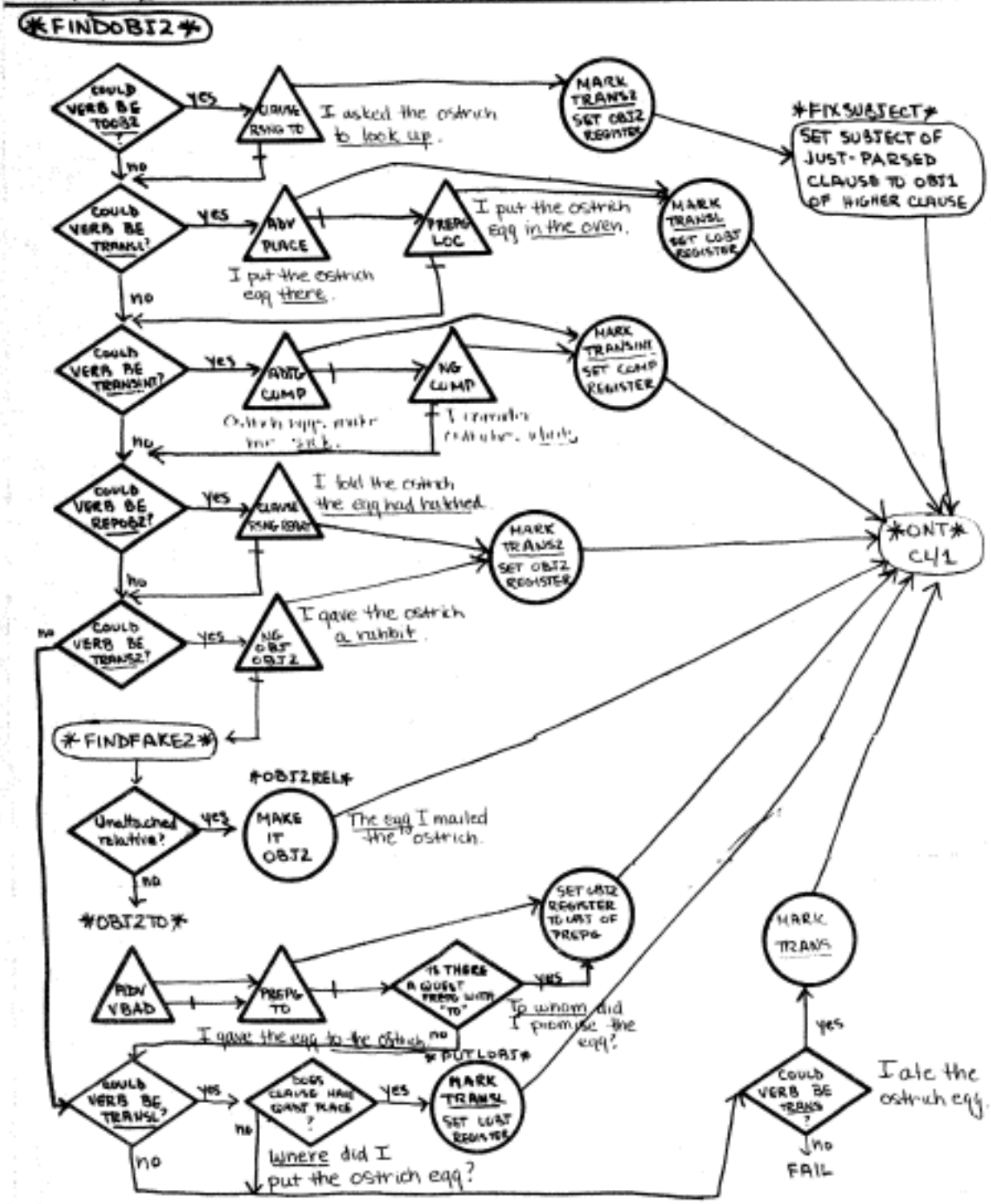


These flowcharts were drawn by me.

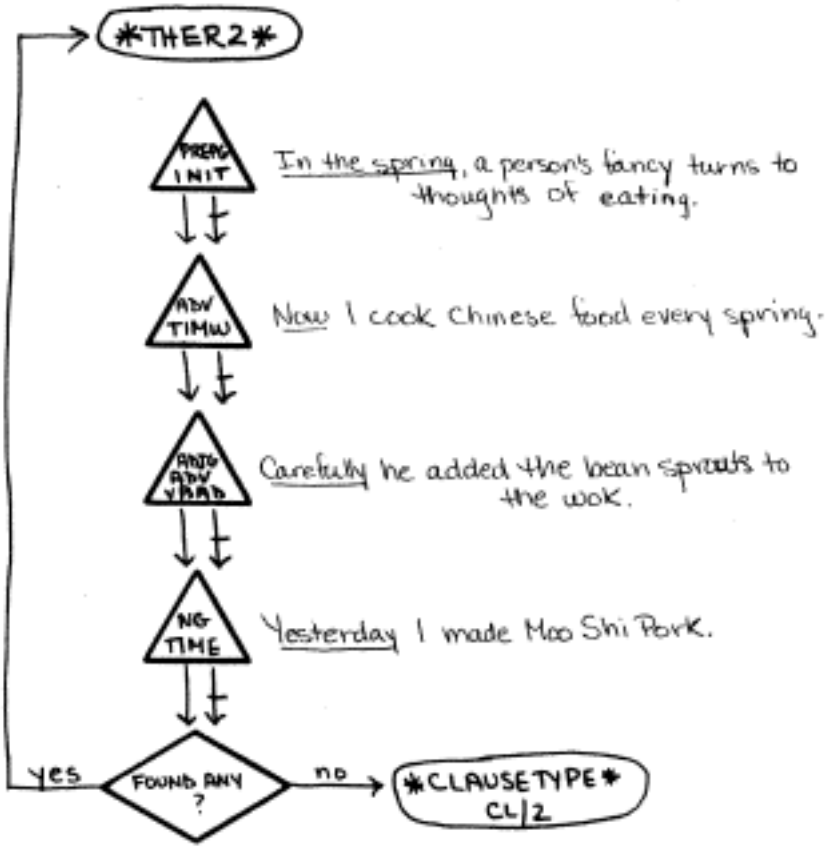
FIND FIRST OBJECT



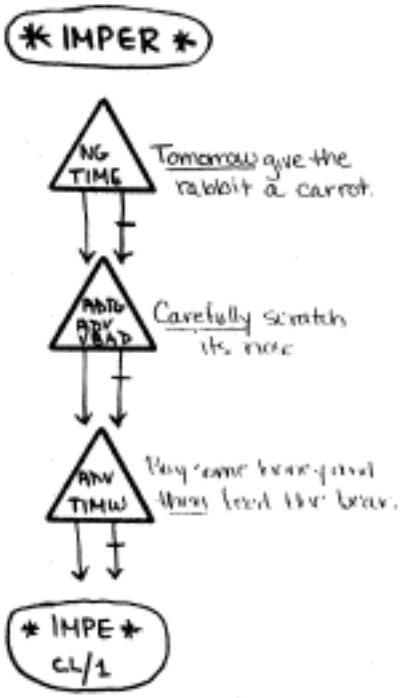
FIND SECOND OBJECT



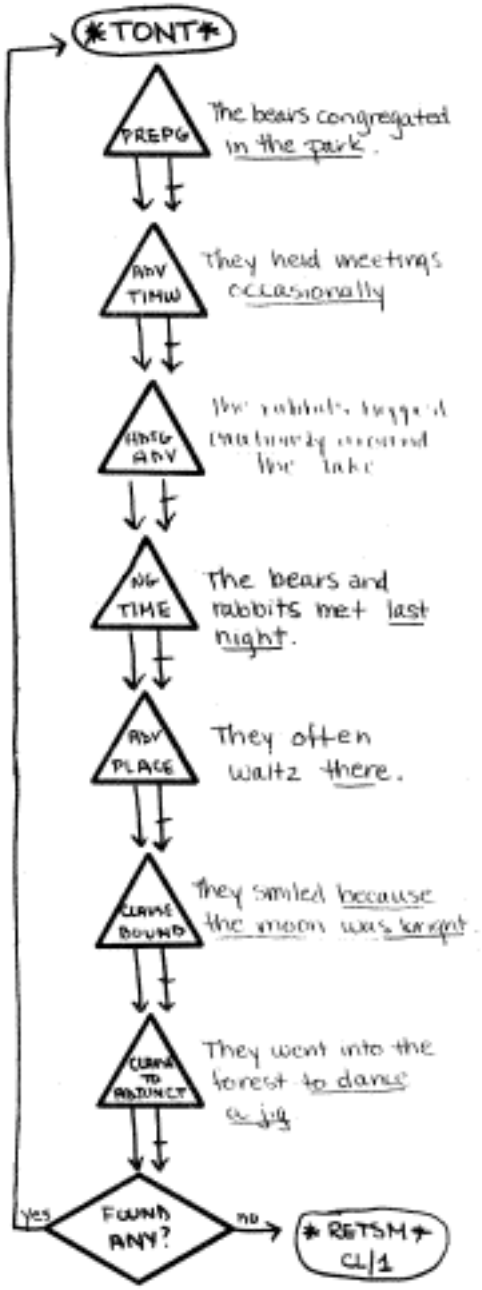
PARSE MODIFIERS (PRE-SUBJECT)



PARSE PRE-VERBAL MODIFIERS

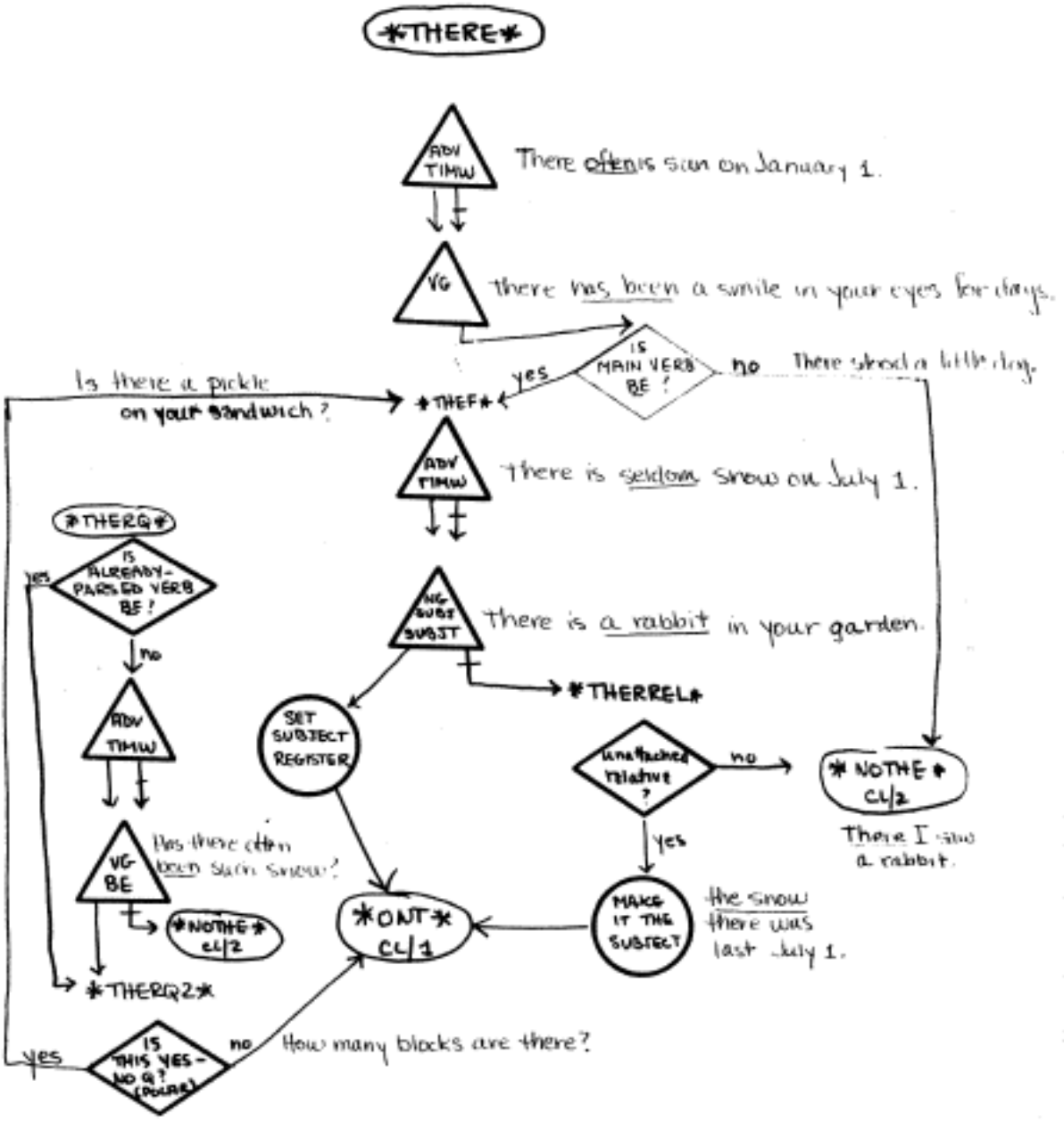


PARSE MODIFIERS (POST-CLAUSE) CL/0



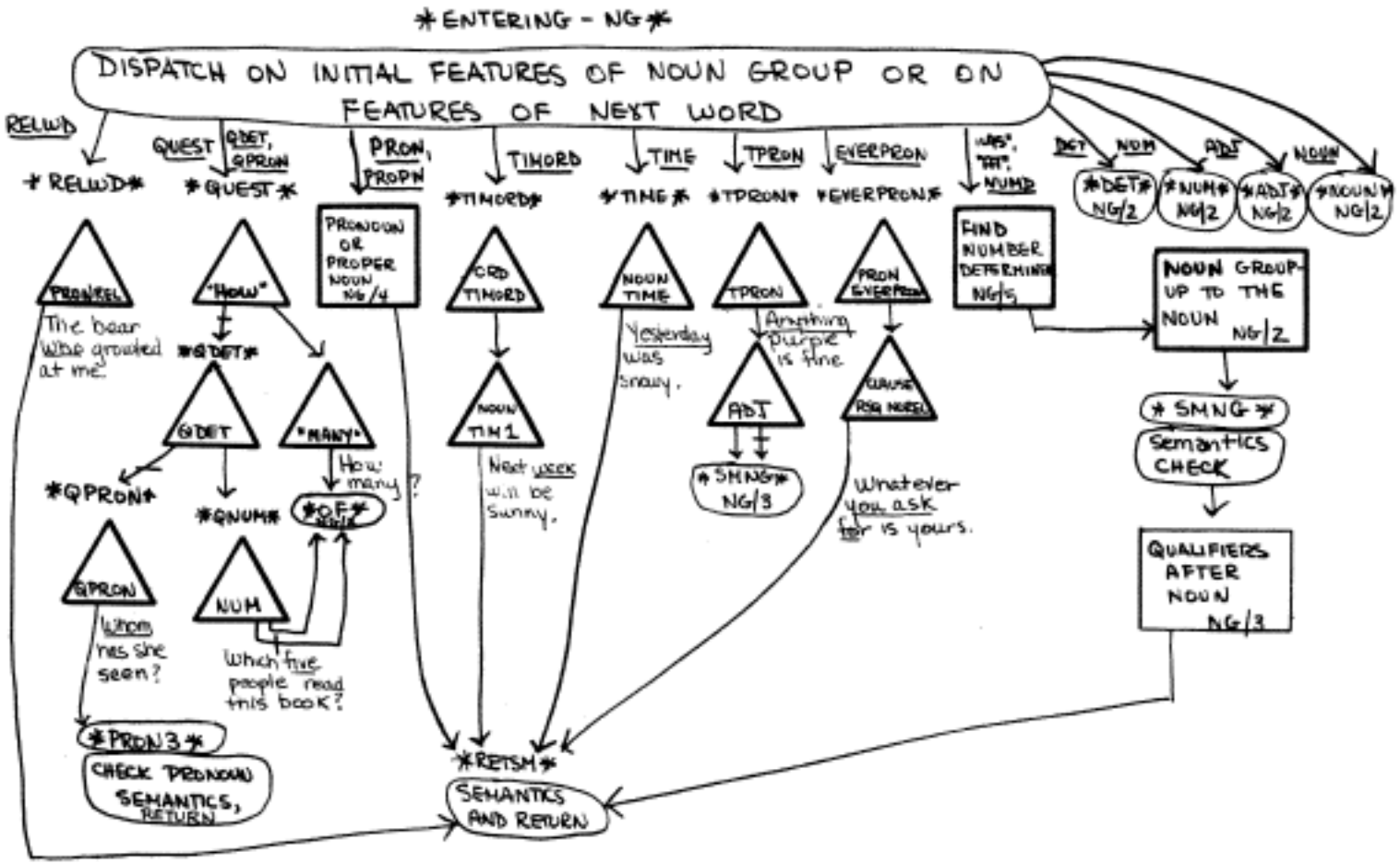
PARSE THERE CONSTITUENTS

cl/11

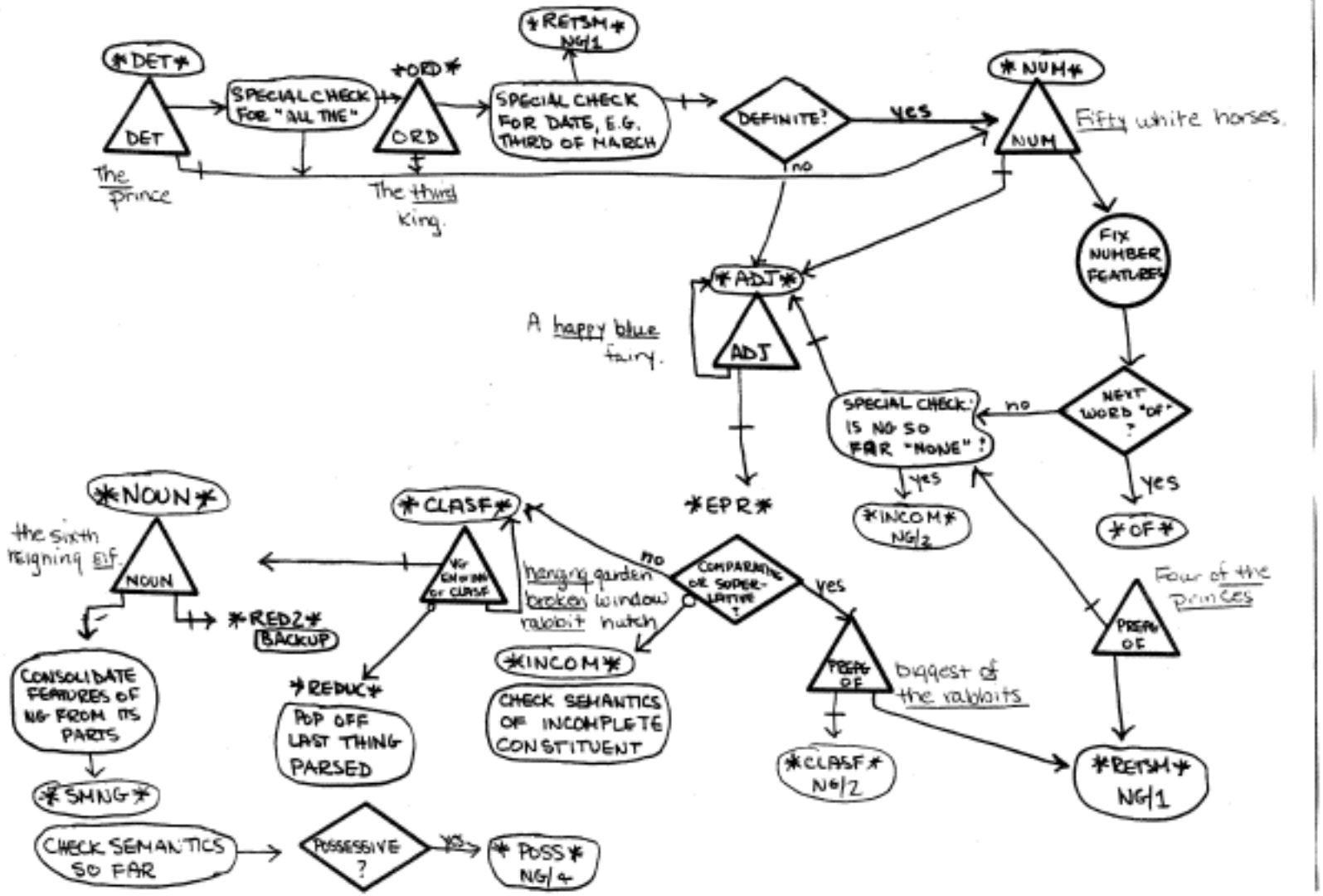


START TO PARSE A NOUN GROUP

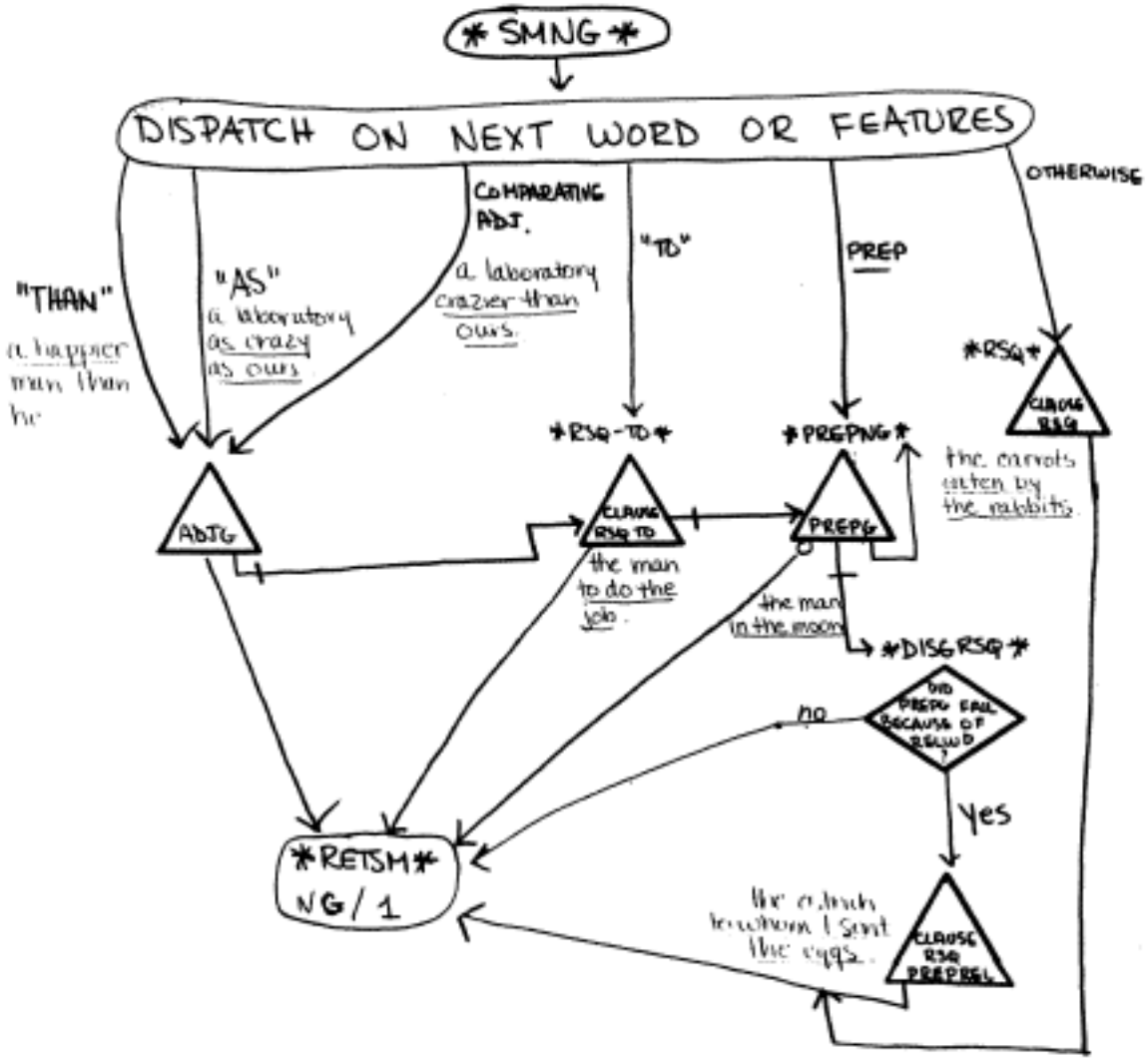
NG/1



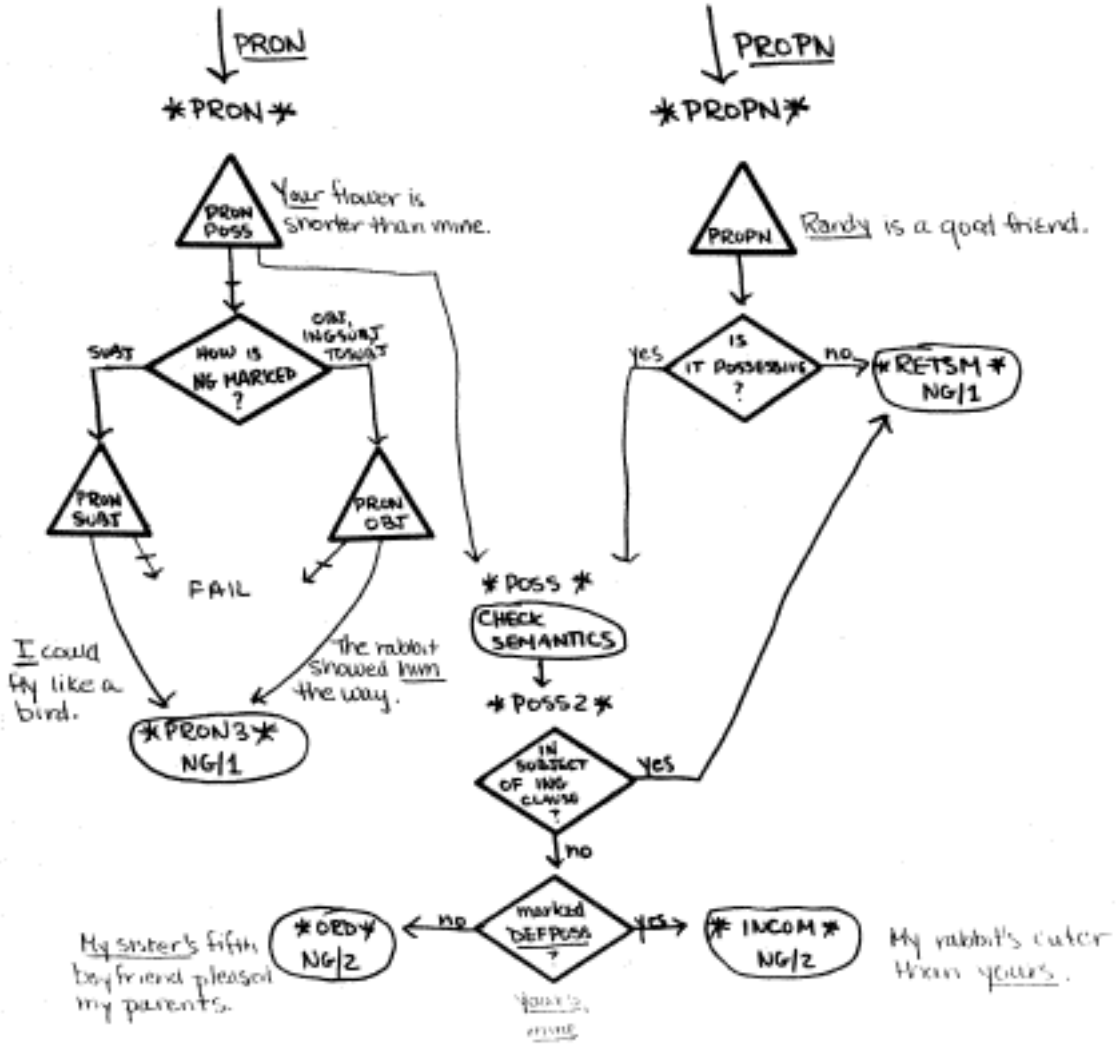
NOUN GROUP - UP TO THE NOUN



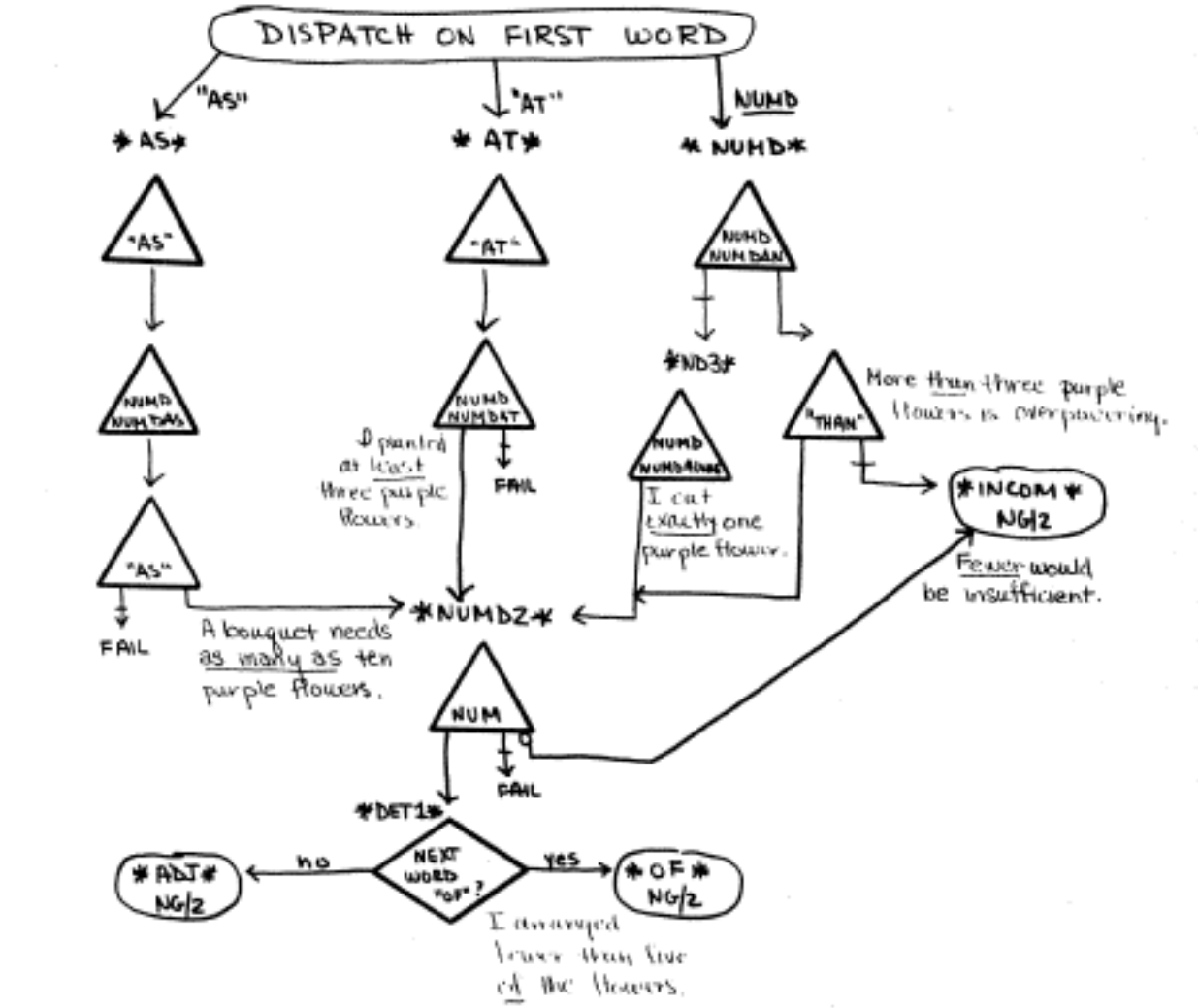
QUALIFIERS AFTER NOUN



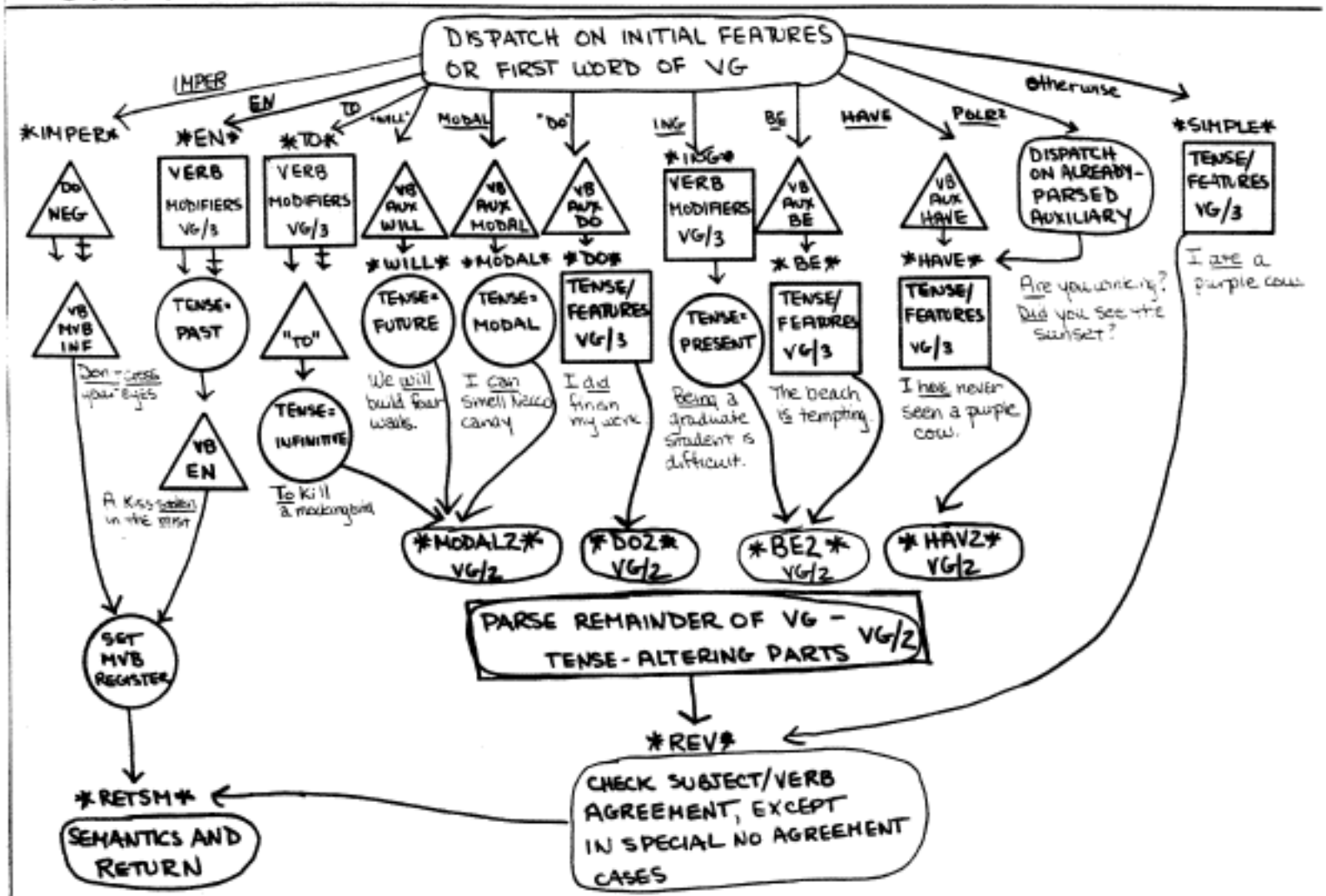
PRONOUN OR PROPER NOUN



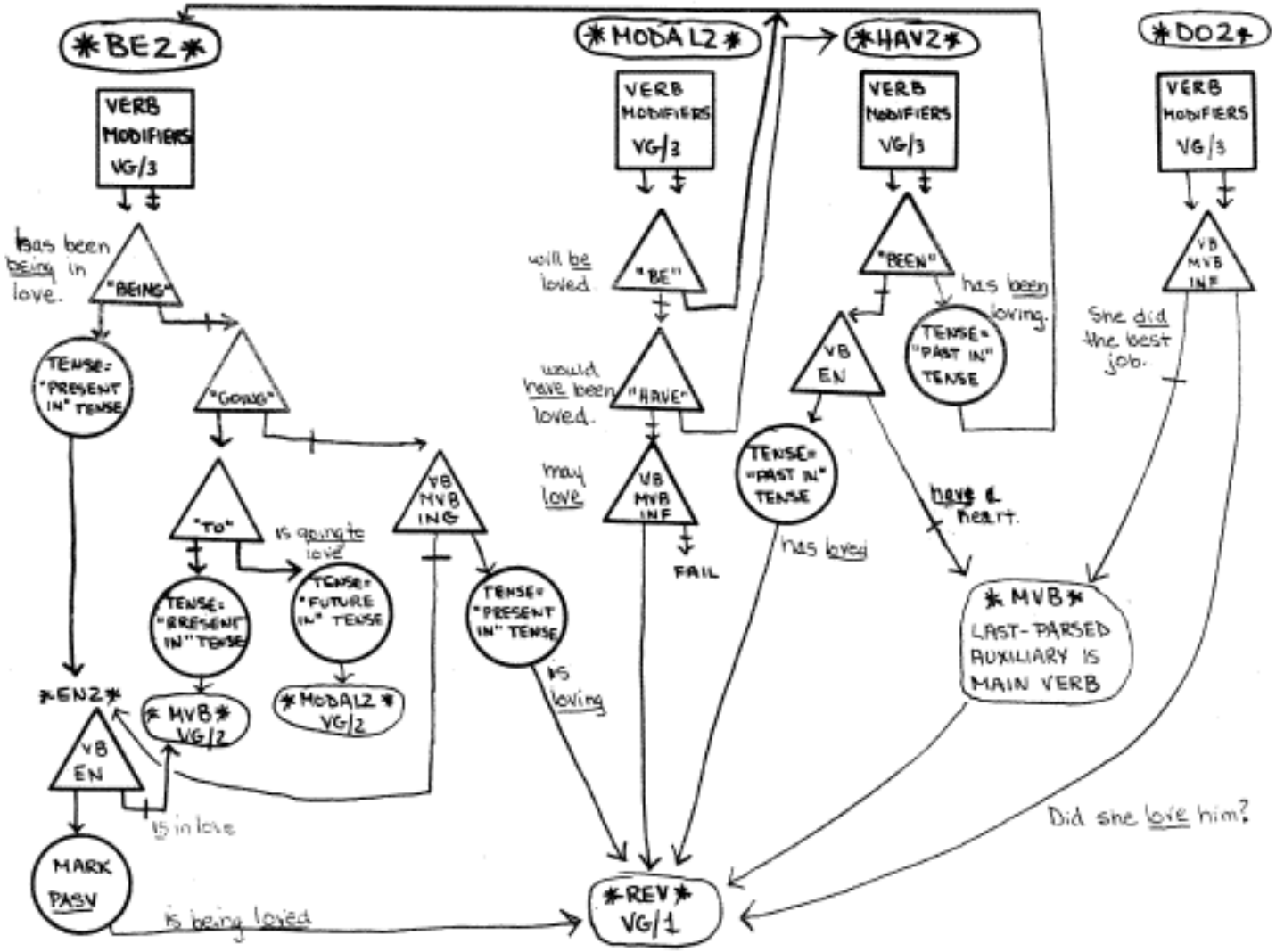
FIND NUMERICAL DETERMINER



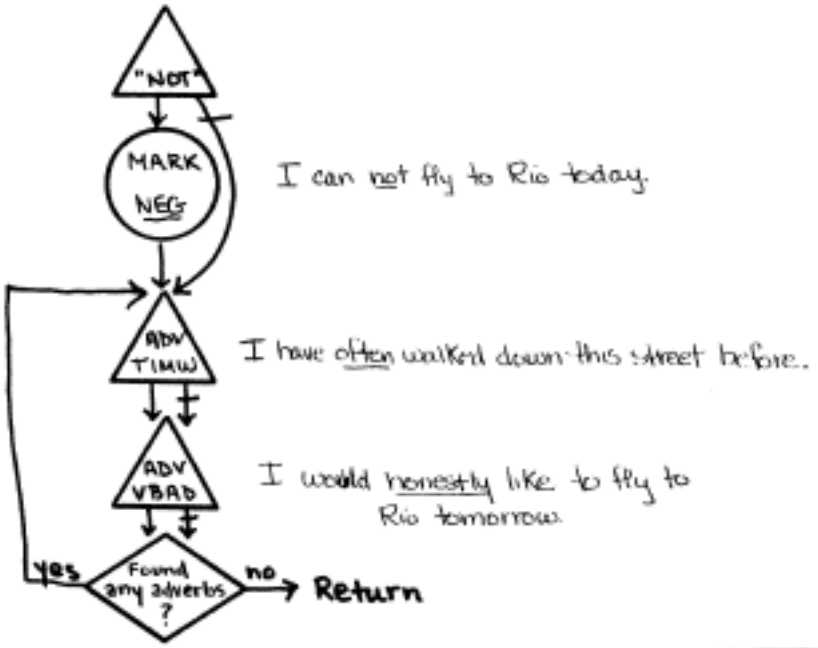
START A VERB GROUP - NUMBER-CARRYING PART



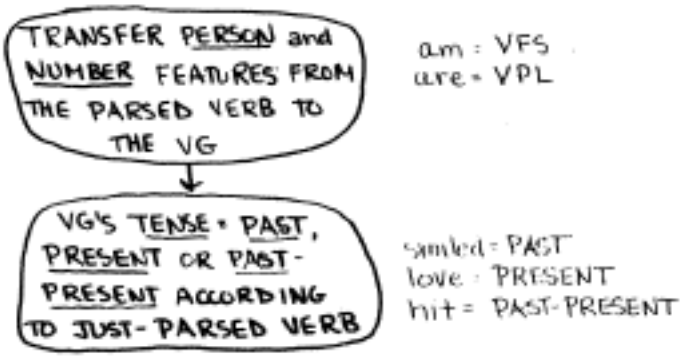
PARSE REMAINDER OF VG - TENSE-ALTERING PARTS



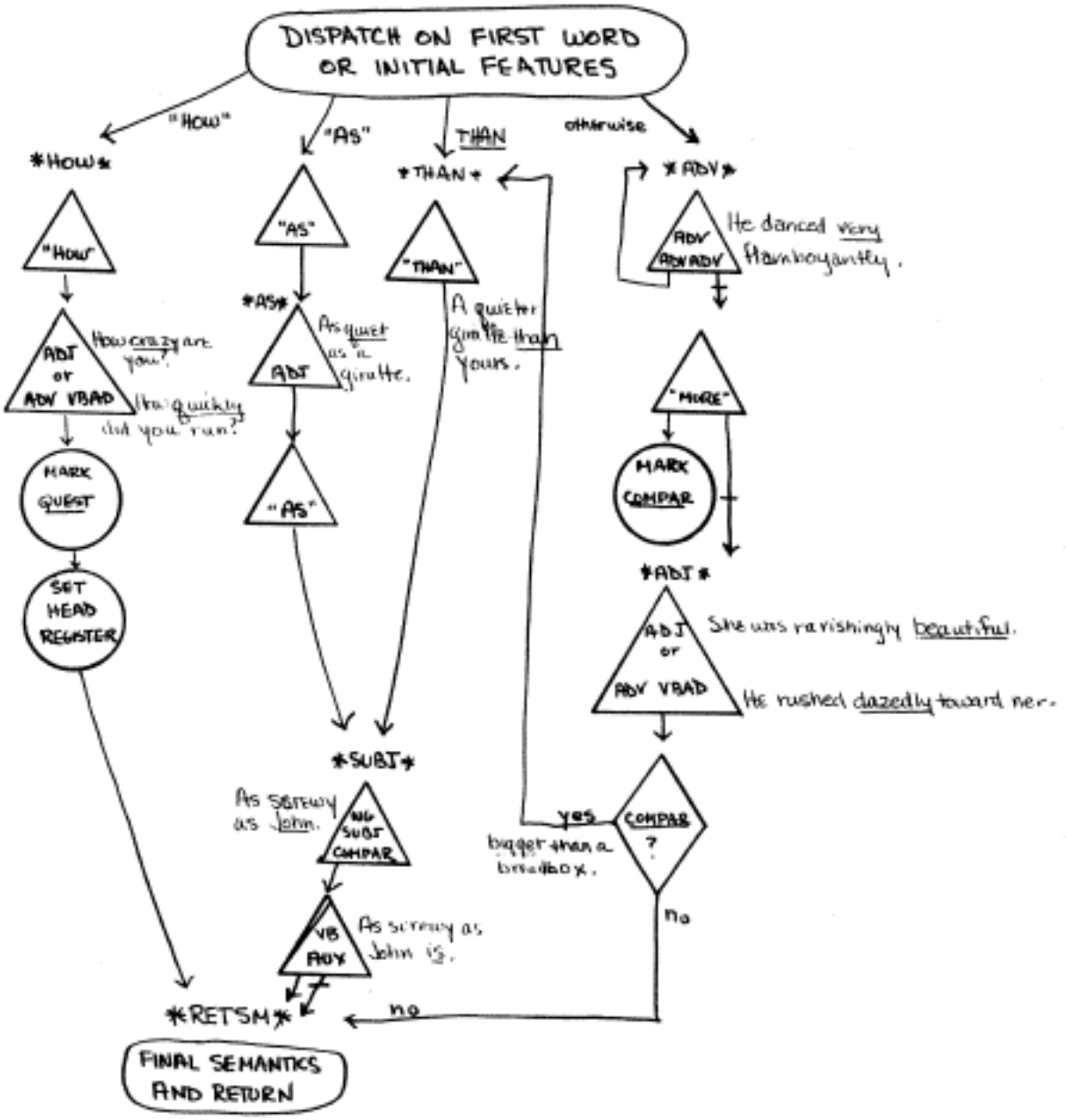
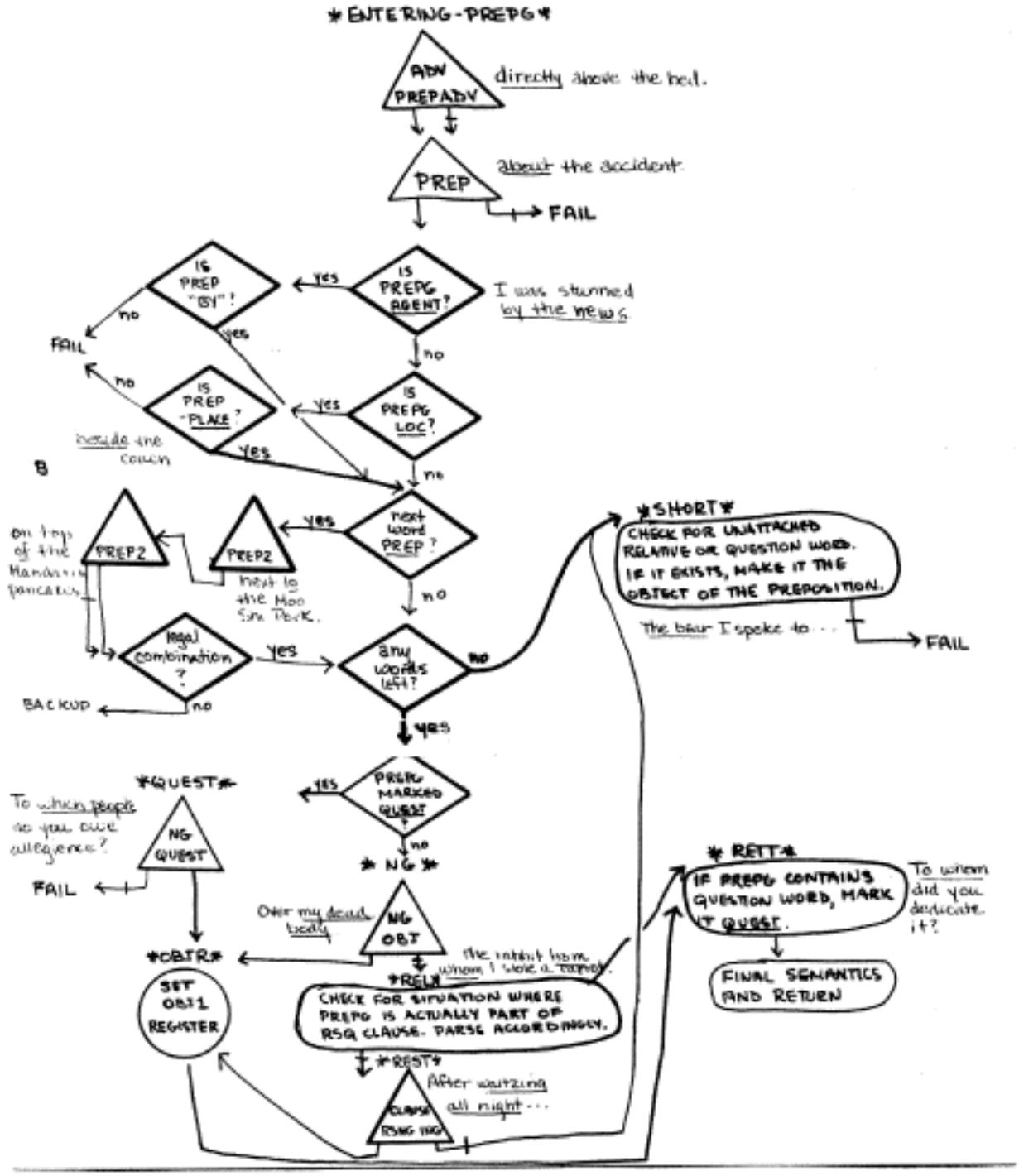
VERB MODIFIERS



TENSE / FEATURES



PARSE ADJECTIVE GROUP



PARSE PREPOSITION GROUP