THE SYNTAX AND SEMANTICS OF USER-DEFINED MODIFIERS IN A TRANSPORTABLE NATURAL LANGUAGE PROCESSOR

Bruce W. Ballard
Dept. of Computer Science
Duke University
Durham, N.C. 27706

ABSTRACT

The Layered Domain Class system (LDC) is an experimental natural language processor being developed at Duke University which reached the prototype stage in May of 1983. Its primary goals are (1) to provide English-language retrieval capabilities for structured but unnormalized data files created by the user; (2) to allow very complex semantics, in terms of the information directly available from the physical data file; and (3) to enable users to customize the system to operate with new types of data. In this paper we shall discuss (a) the types of modifiers LDC provides for; (b) how information about the syntax and semantics of modifiers is obtained from users; and (c) how this information is used to process English inputs.

I INTRODUCTION

The Layered Domain Class system (LDC) is an experimental natural language processor being developed at Duke University. In this paper we concentrate on the types of modifiers provided by LDC and the methods by which the system acquires information about the syntax and semantics of user-defined modifiers. A more complete description is available in [4,5], and further details on matters not discussed in this paper can be found in [1,2,6,8,9].

The LDC system is made up of two primary components. First, the *knowledge acquisition* component, whose job is to find out about the vocabulary and semantics of the language to be used for a new domain, then inquire about the composition of the underlying input file. Second, the *User-Phase Processor*, which enables a user to obtain statistical reductions on his or her data by typed English inputs. The top-level design of the User-Phase processor involves a linear sequence of modules for *scanning* the input and looking up each token in the dictionary; *parsing* the scanned input to determine its syntactic structure; *translation* of the parsed input into an appropriate formal query; and finally *query processing*.

This research has been supported in part by the National Science Foundation, Grants MCS-81-16607 and IST-83-01994; in part by the National Library of Medicine, Grant LM-07003; and in part by the Air Force Office of Scientific Research, Grant 81-0221.

The User-Phase portion of LDC resembles familiar natural language database query systems such as INTELLECT, JETS, LADDER, LUNAR, PHLIQA, PLANES, REL, RENDEZVOUS, TQA, and USL (see [10-23]) while the overall LDC system is similar in its objectives to more recent systems such as ASK, CONSUL, IRUS, and TEAM (see [24-31]).

At the time of this writing, LDC has been completely customized for two fairly complex domains, from which examples are drawn in the remainder of the paper, and several simpler ones. The complex domains are a ***final grades*** domain, giving course grades for students in an academic department, and a ***building organization*** domain, containing information on the floors, wings, corridors, occupants, and so forth for one or more buildings. Among the simpler domains LDC has been customized for are files giving employee information and stock market quotations.

II MODIFIER TYPES PROVIDED FOR

As shown in [4], LDC handles inputs about as complicated as

students who were given a passing grade by an instructor Jim took a graduate course from

As suggested here, most of the syntactic and semantic sophistication of inputs to LDC are due to noun phrase modifiers, including a fairly broad coverage of relative clauses. For example, if LDC is told that "students take courses from instructors", it will accept such relative clause forms as

students who took a graduate course from Trivedi
courses Sarah took from Rogers
instructors Jim took a graduate course from
courses that were taken by Jim
students who did not take a course from Rosenberg

We summarize the modifier types distinguished by LDC in Table 1, which is divided into four parts roughly corresponding to pre-nominal, nominal, post-nominal, and negating modifiers. We have included several modifier types, most of them anaphoric, which are processed syntactically, and methods for whose semantic processing are being implemented along the lines suggested in [7].

Most of the names we give to modifier types are self-explanatory, but the reader will notice that we have chosen to categorize verbs, based upon their semantics, as *trivial* verbs, *implied parameter* verbs, and *operational* verbs. "Trivial" verbs, which involve no semantics to speak of, can be roughly paraphrased as "be associated with". For example, students who *take* a certain course are precisely those students *associated with* the database records related to the course. "Implied parameter" verbs can be paraphrased as a longer "trivial" verb phrase by adding a parameter and requisite noise words for syntactic acceptability. For example, students who *fail* a course are those students who *make a grade of F in* the course. Finally, "operational" verbs require an operation to be performed on one or more of its noun phrase arguments, rather than simply asking for a comparison of its noun phrase referent(s) against values in specified fields of the physical data file. For example, the students who *outscore* Jim are precisely those students who *make a grade higher than the grade of* Jim. At present, prepositions are treated semantically as trivial verbs, so that "students in AI" is interpreted as "students associated with records related to the AI course".

Table 1 - Modifier Types Available in LDC

Modifier Type	Example Usage	Syntax Implemented	Semantics Implemented
Ordinal	the second floor	yes	yes
Superlative	the largest office	yes	yes
Anaphoric Comparative	better students more desirable instructors	yes	no
Adjective	the large rooms classes that were small	yes	yes
Anaphoric Argument-Taking Adjective	adjacent offices	yes	no
Anaphoric Implied-Parameter Verb	failing students	yes	no
Noun Modifier	conference rooms	yes	yes
Subtype	offices	yes	yes
Argument-Taking Noun	classmates of Jim Jim's classmates	yes	yes
Anaphoric Argument-Taking Noun	the best classmate	yes	no
Prepositional Phrase	students in CPS215	yes	(yes)
Comparative Phrase	students better than Jim a higher grade than a C	yes	yes
Trivial Verb Phrase	instructors who teach AI students who took AI from Smith	yes	yes
Implied-Parameter Verb Phrase	students who failed AI	yes	yes
Operational Verb Phrase	students who outscored Jim	yes	yes
Argument-Taking Adjective	offices adjacent to X-238	yes	yes
Negations (of many sorts)	the non graduate students offices not adjacent to X-238 instructors that did not teach AI etc.	yes	yes

III KNOWLEDGE ACQUISITION FOR MODIFIERS

The job of the knowledge acquisition module of LDC, called "Prep" in Figure 1, is to find out about (a) the vocabulary of the new domain and (b) the composition of the physical data file. This paper is concerned only with vocabulary acquisition, which occurs in three stages. In Stage 1, Prep asks the user to name each *entity*, or conceptual data item, of the domain. As each entity name is given, Prep asks for several simple kinds of information, as in

```
ENTITY NAME? section
SYNONYMS: class
TYPE (PERSON, NUMBER, LIST, PATTERN, NONE)?
  pattern
GIVE 2 OR 3 EXAMPLE NAMES: cps51.12, ee34.1
NOUN SUBTYPES: none
ADJECTIVES: large, small
NOUN MODIFIERS: none
HIGHER LEVEL ENTITIES: class
LOWER LEVEL ENTITIES: student, instructor
MULTIPLE ENTITY? yes
ORDERED ENTITY? yes
```

Prep next determines the case structure of verbs having the given entity as surface subject, as in

```
ACQUIRING VERBS FOR STUDENT:
A STUDENT CAN   pass a course
                fail a course
                take a course from an instructor
                make a grade from an instructor
                make a grade in a course
```

In Stage 2, Prep learns the *morhological variants* of words not known to it, e.g. plurals for nouns, comparative and superlative forms for adjectives, and past tense and participle forms for verbs. For example,

```
PAST-TENSE VERB ACQUISITION
PLEASE GIVE CORRECTED FORMS, OR HIT RETURN
  FAIL   FAILED   >
  BITE   BITED    > bit
  TRY    TRIED    >
```

In Stage 3, Prep acquires the *semantics* of adjectives, verbs, and other modifier types, based upon the following principles.

1. Systems which attempt to acquire *complex* semantics from *relatively untrained users* had better restrict the class of the domains they seek to provide an interface to.

For this reason, LDC restricts itself to a class of domains [1] in which the important relationships among domain entities involve hierarchical decompositions.

2. There need not be any correlation between the *type* of modifier being defined and the way in which its *meaning* relates to the underlying data file.

For this reason, Prep acquires the meanings of all user-defined modifiers in the same manner by providing such primitives as *id*, the identity function; *val*, which retrieves a specified field of a record; *num*, which returns the size of its argument, which is assumed to be a set; *sum*, which returns the sum of its list of inputs; *avg*, which returns the average of its list of inputs; and *pct*, which returns the percentage of its list of boolean arguments which are true. Other user-defined adjectives may also be used. Thus, a "desirable instructor" might be defined as an instructor who gave a good grade to more than half his students, where a "good grade" is defined as a grade of B or above. These two adjectives may be specified as shown below.

```
ACQUIRING SEMANTICS FOR DESIRABLE INSTRUCTOR
   PRIMARY?      section
   TARGET?       grade
   PATH IS:      GRADE    /STUDENT    /SECTION
   FUNCTIONS?    good     /id         /pct
   PREDICATE?    > 50

ACQUIRING SEMANTICS FOR GOOD GRADE
   PRIMARY?      grade
   TARGET?       grade
   PATH IS:      GRADE
   FUNCTIONS?    val
   PREDICATE?    >= B
```

As shown here, Prep requests three pieces of information for each adjective-entity pair, namely (1) the *primary* (highest-level) and *target* (lowest-level) entities needed to specify the desired adjective meaning; (2) a *list of functions* corresponding to the arcs on the path from the primary to the target nodes; and finally (3) a *predicate* to be applied to the numerical value obtained from the series of function calls just acquired.

IV UTILIZATION OF THE INFORMATION ACQUIRED DURING PREPROCESSING

As shown in Figure 1, the English-language processor of LDC achieves domain independence by restricting itself to (a) a domain-independent, linguistically-motivated phrase-structure grammar [6] and (b) and the domain-specific files produced by the knowledge acquisition module.

The simplest file is the *pattern* file, which captures the morphology of domain-specific proper nouns, e.g. the entity type "room" may have values such as X-238 and A-22, or "letter, dash, digits". This information frees us from having to store all possible field values in the dictionary, as some systems do, or to make reference to the physical data file when new data values are typed by the user, as other systems do.

The domain-specific *dictionary* file contains some standard terms (articles, ordinals, etc.) and also both root words and inflections for terms acquired from the user. The sample dictionary entry

(longest Superl long (nt meeting week))

says that "longest" is the superlative form of the adjective "long", and may occur in noun phrases whose head noun refers to entities of type meeting or week. By having this information in the dictionary, the parser can perform "local" compatibility checks to assure the

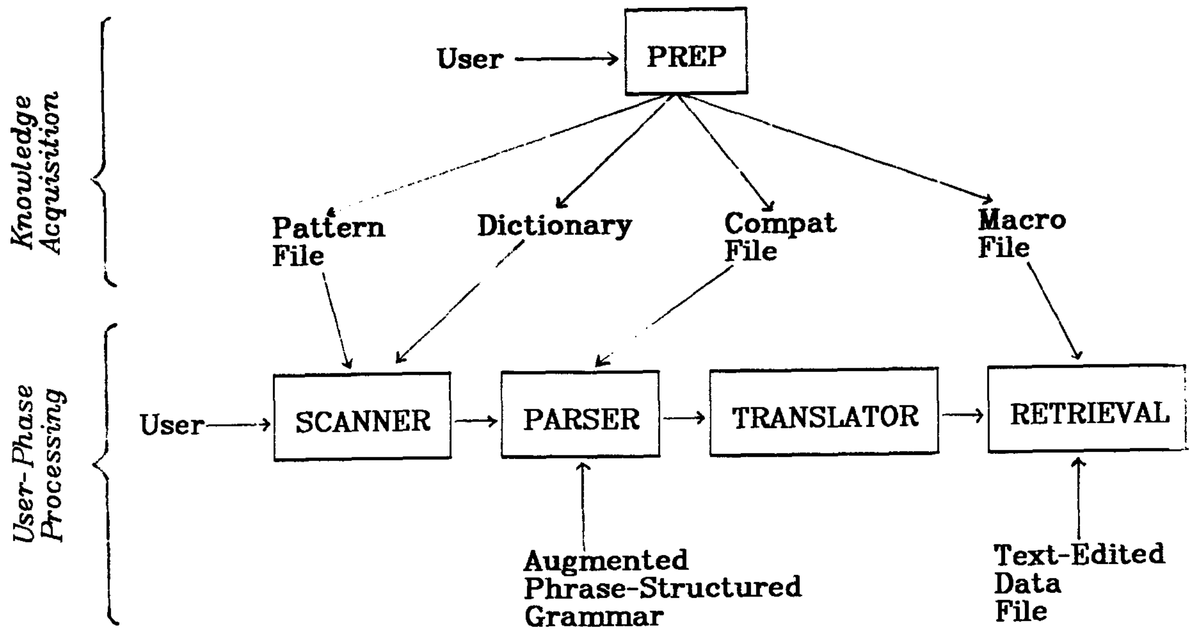

Figure 1 - Overview of LDC

integrity of a noun phrase being built up, i.e. to assure all words in the phrase can go together on non-syntactic grounds. This aids in disambiguation, yet avoids expensive interaction with a subsequent semantics module.

An opportunity to perform "non-local" compatibility checking is provided for by the *compat* file, which tells (a) the case structure of each verb, i.e. which prepositions may occur and which entity types may fill each noun phrase "slot", and (b) which pairs of entity types may be linked by each preposition. The former information will have been acquired directly from the user, while the latter is predicted by heuristics based upon the sorts of conceptual relationships that can occur in the "layered" domains of interest [1].

Finally, the *macro* file contains the meanings of modifiers, roughly in the form in which they were acquired using the specification language discussed in the previous section. Although this required us to formulate our own retrieval query language [3], having complex modifier meanings directly exceutable by the retrieval module enables us to avoid many of the problems typically arising in the translation from parse structures to formal retrieval queries. Furthermore, some modifier meanings can be *derived* by the system from the meanings of other modifiers, rather than separately acquired from the user. For example, if the meaning of the adjective "large" has been given by the user, the system automatically processes "largest" and "larger than ..." by appropriately interpreting the macro body for "large".

A partially unsolved problem in macro processing involves the resolution of scope ambiguities related to negation Interestingly, most meaningful interpretations of phrases containing "non" or "not" can be obtained by inserting the retrieval module's Not command at an appropriate point in the macro body for the modifier in question. For example,

students who were not failed by Rosenberg

might or might not be intended to include students who did not take a course from Rosenberg. The retrieval query commands generated by the positive usage of "fail", as in

students that Rosenberg failed

would be the sequence

```
instructor = Rosenberg;
student -> fail
```

so the question is whether to introduce "not" at the phrase level

```
not {instructor = Rosenberg;
     student -> fail}
```

or instead at the verb level

```
instructor = Rosenberg;
not {student -> fail}
```

Our current system takes the literal reading, and thus generates the first interpretation given. The example points out the close relationship between negation scope and the important problem of "presupposition", in that the user may be interested only in students who had a chance to be failed.

REFERENCES

1. Ballard, B. A "Domain Class" approach to transportable natural language processing. *Cognition and Brain Theory*, 5 (1982), 3, pp. 269-287.

2. Ballard, B. and Lusth, J. An English-language processing system that "learns" about new domains. *AFIPS National Computer Conference*, 1983, pp. 39-46.

3. Ballard, B. and Lusth, J. The design of DOMINO: a knowledge-based information retrieval processor for office enviroments. Tech. Report CS-1984-2, Dept. of Computer Science, Duke University, February 1984.

4. Ballard, B., Lusth, J. and Tinkham, N. LDC-1: a transportable, knowledge-based natural language processor for office environments. *ACM Trans. on Office Information Systems*, 2 (1984), 1, pp. 1-25.

5. Ballard, B., Lusth, J. and Tinkham, N. Transportable English language processing for office environments. *AFIPS National Computer Conference*, 1984, to appear in the proceedings.

6. Ballard, B. and Tinkham, N. A phrase-structured grammatical formalism for transportable natural language processing. *Amer. J. Computational Linguistics*, to appear.

7. Biermann, A. and Ballard, B. Toward natural language computation. *Amer. J. Computational Linguistics*, 6 (1980), 2, pp. 71-86.

8. Lusth, J. Conceptual Information Retrieval for Improved Natural Language Processing (Master's Thesis). Dept. of Computer Science, Duke University, February 1984.

9. Lusth, J. and Ballard, B. Knowledge acquisition for a natural language processor. *Conference on Artificial Intelligence*, Oakland University, Rochester, Michigan, April 1983, to appear in the proceedings.

10. Bronnenberg, W., Landsbergen, S., Scha, R., Schoenmakers, W. and van Utteren, E. PHLIQA-1, a question-answering system for data-base consultation in natural English. *Philips tech. Rev.* 38 (1978-79), pp. 229-239 and 269-284.

11. Codd, T. Seven steps to RENDEZVOUS with the casual user. In *Data Base Management*, J. Kimbie and K. Koffeman (Eds.), North-Holland, 1974.

12. Codd, T. RENDEZVOUS Version 1: An experimental English-language query formulation system for casual users of relational data bases. IBM Research Report RJ2144, San Jose, Ca., 1978.

13. Finin, T., Goodman, B. and Tennant, H. JETS: achieving completeness through coverage and closure. *Int. J. Conf. on Artificial Intelligence*, 1979, pp. 275-281.

14. Harris, L. User-oriented data base query with the Robot natural language system. *Int. J. Man-Machine Studies*, 9 (1977), pp. 697-713.

15. Harris, L. The ROBOT system: natural language processing applied to data base query. *ACM National Conference*, 1978, pp. 165-172.

16. Hendrix, G. Human engineering for applied natural language processing. *Int. J. Conf. on Artificial Intelligence*, 1977, pp. 183-191.

17. Hendrix, G., Sacerdoti, E., Sagalowicz, D. and Slocum, J. Developing a natural language interface to complex data. *ACM Trans. on Database Systems*, 3 (1978), 2, pp. 105-147.

18. Lehmann, H. Interpretation of natural language in an information system. *IBM J. Res. Dev.* 22 (1978), 5, pp. 560-571.

19. Plath, W. REQUEST: a natural language question-answering system. *IBM J. Res. Dev.*, 20 (1976), 4, pp. 326-335.

20. Thompson, F. and Thompson, B. Practical natural language processing: the REL system as prototype. In *Advances in Computers*, Vol. 3, M. Rubinoff and M. Yovits, Eds., Academic Press, 1975.

21. Waltz, D. An English language question answering system for a large relational database. *Comm. ACM* 21 (1978), 7, pp. 526-539.

22. Woods, W. Semantics and quantification in natural language question answering. In *Advances in Computers*, Vol. 17, M. Yovits, Ed., Academic Press, 1978.

23. Woods, W., Kaplan, R. and Nash-Webber, B. *The Lunar Sciences Natural Language Information System: Final Report*. Report 2378, Bolt, Beranek and Newman, Cambridge, Mass., 1972.

24. Ginsparg, J. A robust portable natural language data base interface. *Conf. on Applied Natural Language Processing*, Santa Monica, Ca., 1983, pp. 25-30.

25. Grosz, B. TEAM: A transportable natural language interface system. *Conf. on Applied Natural Language Processing*, Santa Monica, Ca., 1983, pp. 39-45.

26. Haas, N. and Hendrix, G. An approach to acquiring and applying knowledge. *First Nat. Conf. on Artificial Intelligence*, Stanford Univ., Palo Alto, Ca., 1980, pp. 235-239.

27. Hendrix, G. and Lewis, W. Transportable natural-language interfaces to databases. *Proc. 19th Annual Meeting of the ACL*, Stanford Univ., 1981, pp. 159-165.

28. Mark, W. Representation and inference in the Consul system. *Int. Joint Conf. on Artificial Intelligence*, 1981.

29. Thompson, B. and Thompson, F. Introducing ASK, a simple knowledgeable system. *Conf. on Applied Natural Language Processing*, Santa Monica, Ca., 1983, pp. 17-24.

30. Thompson, F. and Thompson, B. Shifting to a higher gear in a natural language system. *National Computer Conference*, 1981, 657-662.

31. Wilczynski, D. Knowledge acquisition in the Consul system. *Int. Joint Conf. on Artificial Intelligence*, 1981.