MAPPING BETWEEN SEMANTIC REPRESENTATIONS USING HORN CLAUSES[1]

Ralph M. Weischedel
Computer & Information Sciences
University of Delaware
Newark, DE 19711

ABSTRACT

Even after an unambiguous semantic interpretation has been computed for a sentence in context, there are at least three reasons that a system may map the semantic representation R into another form S.

1. The terms of R, while reflecting the user view, may require deeper understanding, e.g. may require a version S where metaphors have been analyzed.

2. Transformations of R may be more appropriate for the underlying application system, e.g. S may be a more nearly optimal form. These transformations may not be linguisticly motivated.

3. Some transformations may depend on non-structural context.

Design considerations may favor factoring the process into two stages, for reasons of understandability or for easier transportability of the components.

This paper describes the use of Horn clauses for the three classes of transformations listed above. The transformations are part of a system that converts the English description of a software module into a formal specification, i.e. an abstract data type.

[1] RESEARCH SPONSORED BY THE AIR FORCE OFFICE OF SCIENTIFIC RESEARCH, AIR FORCE SYSTEM COMMAND, USAF, UNDER GRANT NUMBER AFOSR-80-0190C. THE UNITED STATES GOVERNMENT IS AUTHORIZED TO REPRODUCE AND DISTRIBUTE REPRINTS FOR GOVERNMENTAL PUPOSES NOTWITHSTANDING ANY COPYRIGHT NOTATION HEREIN.

1. INTRODUCTION

Parsing, semantic interpretation, definite reference resolution, quantifier scope decisions, and determining the intent of a speaker/author are well-known problems of natural langauge understanding. Yet, even after a system has generated a semantic representation R where such decisions have been made, there may still be a need for further transformation and understanding of the input to generate a representation S for the underlying application system. There are at least three reasons for this.

First, consider spatial metaphor. Understanding spatial metaphor seems to require computing some concrete interpretation S for the metaphor; however, understanding the metaphor concretely may be attempted after computing a semantic representation R that represents the spatial metaphor formally but without full understanding. Explaining the system's interpretation of a user input (e.g. for clarification dialog, allowing the user to check the system's undersanding, etc.) is likely to be more understandable if the terminology of the user is employed. By having an intermediate level of understanding such as R, and generating English output from it, one may not have to recreate the metaphor, for the terms in R use it as a primitive.

Second, the needs of the underlying application system may dictate transformations that are neither essential to understanding the English text nor linguisticly motivated. In a data base environment, transformations of the semantic representation may yield a retrieval request that is computationally less demanding [11]. To promote portability, EUFID [13] and TQA [6] are interfaces that have a separate component for transformations specific to the data base. In software specification, mapping of the semantic representation R may yield a form S which is more amenable for proving theorems about the specification or for rewriting it into some standard form.

The following example, derived from a definition of stacks on page 77 of [10] illustrates both of the reasons above.

> A stack is an ordered list in which all insertions and deletions occur at one end called the top.

A theorem prover for abstract data types would normally assume that the end of the stack in question is referred to by a notation such as A[1] if A is the name of the stack, rather than understanding the spatial metaphor "one end".

Third, it may be convenient to design the transformation process in two phases where the output of both phases is a semantic representation. In our system, we have chosen to map certain paraphrases into a common form via a two step process. The forms "ith element" and "element i" each generate the same term as a result of semantic interpretation. However, the semantic interpreter generates another term for "element at position i" due to the extra lexical items "at" and "position". Obviously, all three expressions correspond to one concept. The mapping component recognizes that the two terms generated by the semantic interpreter are paraphrases and maps them into one form.

Section 2 gives an overview of the system as a whole. Section 3 describes the use of Horn clauses for the mapping from R to S. Related research and our conclusions are presented in sections 4 and 5.

2. BRIEF SYSTEM OVERVIEW

The overall system contains several components beside the mapping component that is the focus of this paper. The system takes as input short English texts such as the data structure descriptions in [10]. The output is a formal specification of the data structure defined in Horn clauses[2].

First, the RUS parser [3], which includes a large general-purpose grammar of English, calls a semantic component to incrementally compute the semantic interpretation of the phrases being proposed. As soon as a phrase is proposed by the grammar, the semantic interpreter either generates a semantic interpretation for the phrase or vetoes the parse. The only modifications to adapt the parser to the application of abstract data types were to add mathematical notation, so that phrases such as "the list (A[1], A[2], ..., A[N])" could be understood. Thus, a text such as the following can be parsed by the modified grammar[3].

---

[2] Horn clauses are a version of first order logic, where all well-formed formulas have the form C IF A1 & A2 & ... & An. Each of the Ai is an atomic formula; C is an atomic formula; and n>=0. Therefore, all variables are free.

[3] This is a modified version of a definition given on pages 41-42 of [10].

1. We say that an ordered list is empty or it can be written as (A[1], A[2], ..., A[N]) where the A[I] are atoms from some set S.

2. There are a variety of operations that are performed on these lists.

3. These operations include the following.

4. Find the length N of the list.

5. Retrieve the ith element, 1<=I<=N.

6. Store a new value at the ith position, 1<=I<=N.

7. Insert a new element at position I, 1<=I<=N+1 causing elements numbered I, I+1, ..., N to become numbered I+1, I+2, ..., N+1.

8. Delete the element at position I, 1<=I<=N causing elements numbered I+1, ..., N to become numbered I, I+1, ., N-1.

The semantic component we developed employs case frames for disambiguation and generation of the semantic interpretation of a phrase. However, the semantic component does not make quantifier scope decisions.

Quantifier scope decisions, reference resolution, and conversion from first-order logic to Horn clauses is performed after the semantic interpreter has completed its processing. The knowledge governing these three tasks is itself encoded in Horn clauses and was developed by Daniel Chester. The output from this component is the input to the mapping component, which is the focus of this paper. In the appendix, examples of the Horn clause input to the mapping component are given for some of the sentences of the text above.

The semantic representation R of a single sentence is therefore a set of Horn clauses. In addition, the model of context built in understanding the text up to the current sentence is a set of Horn clauses and a list of entities which could be referenced in succeeding sentences. The mapping component performs the three tasks described in the previous section to generate a set S of Horn clauses. S is added to the model of context prior to processing the next input sentence.

The choice of Horn clauses as the formal representation of the abstract data type is based on the following motivations:

1. Once a text has been understood, the set of Horn clauses can be added to the knowledge base (which is also encoded as Horn clauses). This offers the potential of a system that grows in power.

2. The semantics of Horn clauses, their use in theorem proving, and their executability makes them an appropriate formalism for defining abstract data types.

3. A Horn clause theorem prover [5] allowing free intermixing of lisp and theorem proving is readily available.

3. MAPPING IN THE SYSTEM

The rules of the mapping component are all encoded as Horn clauses. The antecedent atomic formulas of our rules specify either

1. the structural change to be made in the collection of formulas or

2. conditions which are not structural in nature but which must be true if the mapping is to apply.

An underscore preceding an identifier means that the identifier is a variable. We will use the notation (MAPPING-RULE (a1 ... am) __x (c1 ... ck) __y) to mean that the atomic formulas a1 ... am must be present in the list __x of atomic formulas; the list __x of formulas is assumed to be implicitly conjoined. The variable __y will be bound to the result of replacing the formulas a1, ..., am in __x with the formulas c1, ..., ck. There is a map between two lists, __x and __y, of atomic formulas if (MAP __x __y) is true.

Three examples are detailed next. For expository purposes the rules given in this section are simplified.

Consider the following example: *A stack is an ordered list in which all insertions and deletions occur at one end called the top. ADD(I,S) adds item I to stack S.* In this environment spatial metaphors tend to be more frozen than creative. To understand "one end", we assume the following rules:

1. For a sequence __D, we may map "__E is an end of __D" to "__E is the first sequence element of __D".

2. An ordered list is a sequence.

Facts (1) and (2) are encoded as Horn clauses below.

```
1. (MAP __X __Y) IF
     (SEQUENCE __D) &
     (MAPPING-RULE
       ((END __E __D)) __X
       ((SEQUENCE-ELEMENT __E 1 __D)) __Y)

2. (SEQUENCE __D) IF (ORDERED-LIST __D)
```

The system knows how to map the notion of "end of a sequence", and it knows that ordered lists are sequences. Since the first sentence is discussing the end of an ordered list, the two rules above are sufficient to map "end" into the appropriate concrete semantic representation. The power and generality of this approach is that

- a chain of reasoning may show how to view some entity __D as a sequence (and therfore the rules show how to interpret "end of __D"), and

- other mapping rules may state how to interpret spatial metaphors unrelated to "end" or to sequences.

We propose that the same mechanism can deal with certain vague, extended uses of words, such as *add* in the previous example. In stating that ADD(I,S) adds item I to stack S, *add* cannot be predefined for stacks, since its meaning is being defined. Nevertheless, it is reasonable to assume that there is a general relation between *add* and related concepts such as uniting, including, or, in the data structure environments, inserting. Consequently, we propose the following fact in addition to the two above:

- For a sequence __S, we may map "add __I to __S" to "insert __I at some position __X of __S".

It may be stated formally as

```
(MAP __W __Z) IF
  (MAPPING-RULE ((ADD __I __S)) __W
                ((INSERT __I __S __X)) __Z)
  & (SEQUENCE __S)
```

Notice that __X will be unbound. However, the Horn clauses generated for the first sentence (*A stack is an ordered list in which all insertions and deletions occur at one end called the top*) will imply that __X is the position corresponding to the end called top. Therefore, the vague, extended use of "add" can be understood using the inference mechanism of the mapping component. Other rules may state how to interpret an extended use of *add* by relating it to views other than sequences.

Another example involves mapping the forms "ith element", "element i", and "element at position i" into the same representation. Assume that the semantic interpreter generates for each of the first two the list of formulas ((ELEMENT __X) (IDENTIFIED-BY __X __Y)). The Horn clause for that mapping is as follows:

```
(MAP __W __Z) IF
  (SEQUENCE __T) & (TOPIC __T) &
  (MAPPING-RULE
    ((ELEMENT __X) (IDENTIFIED-BY __X __Y))
    __W ((SEQUENCE-ELEMENT __X __Y __T)) __Z)
```

Note that this rule assumes that in context some sequence __T has been identified as the topic; the rule identifies that the element __X is the __Yth member of the sequence __T. For the phrase "element at position i", assume the semantic interpreter generates the list of formulas ((ELEMENT __X) (AT __X (POSITION __P)) (IDENTIFIED-BY __P __Y)). The mapping rule for it is similar to the one above.

```
(MAP __W __Z) IF
  (SEQUENCE __T) & (TOPIC __T) &
  (MAPPING-RULE
    ((ELEMENT __X) (AT __X (POSITION __P))
     (IDENTIFIED-BY __P __Y))
    __W ((SEQUENCE-ELEMENT __X __Y __T)) __Z)
```

This second rule must be tried before the prior one.

The mapping component maps from a representation R of a single sentence to another representation S, given the context of the sentence. For Horn clauses, mapping rules may apply to the antecedents or to the consequents. For each Horn clause, its antecedents are collected in a list and bound to __W in the relation (MAP __W __Z). The result of applying one rule is bound to __Z. Mapping is then tried on the result __Z, and so on until no more rules apply. In additin to applying to antecedents, mapping rules apply to the consequents C1, ..., Ck (k>0) of Horn clauses having the same list of antecedents. C1, ..., Ck are collected in a list and bound to __W in the relation (MAP __W __Z) as for the case of antecedents. The mapper halts when no more rules can be applied. The result of the mapping is a new set of Horn clauses which are the representation S.

4. RELATED WORK

A number of applied AI systems have been developed to support automating software construction [1, 8, 2, 7]. Of these, only our effort has focussed on the mapping problem.

Viewing spatial metaphors in terms of a scale was proposed in [9]. Our model is somewhat more general in that the inference process

- permits specific constraints for each metaphor, not just the one view of a scale, and

- accounts for other mapping problems in addition to spatial metaphor.

A very similar approach to mapping has been proposed in [12]. Instead of using Horn clauses as the formalism for mapping, they encode their rules in KL-ONE [4]. The concern in [12] is inferring the appropriate service to perform in response to a user request, rather than demonstrating means of interpreting spatial metaphors or of finding contextually dependent paraphrases.

5. CONCLUSIONS

There are several reasons why one may have a second transduction phase even after a semantic representation for an utterance has been computed. The advantage of using Horn clauses (or any other deduction mechanism) in this mapping phase is the ability to include nonstructural conditions. This means that the mapping rules may be based on reasoning about the context.

There are three areas for future work:

- generating mapping rules based on additional texts,

- investigating use of the mapping component in reference resolution, and

- developing an indexing technique to run the mapper in a forward chaining mode.

APPENDIX

For sentences 3 through 7 of the example text in section 2, we include here the actual Horn clauses that serve as the output of the semantic component and as the input to the mapping component. The English that generated the Horn clauses is provided for reference in italics; it is not supplied as input to the mapping component. Ampersands have been inserted for expository purposes.

These operations include the following.

```
(((INCLUDE A16 A340)
  IF (FOLLOW A340)
     & (EQUIV A16
         (SETOF A0034
           (AND (OPERATION A0034)
                (PERFORM NIL A0034 A23))))
     & (LIST A23) & (ORDER NIL A23)))
```

Find the length, N, of the list.

```
(((EQUIV (A0037 A23) N)
  IF (LIST A23) & (ORDER NIL A23))
 ((LENGTH (A0038 A23) A23)
  IF (LIST A23) & (ORDER NIL A23))
 (((EQUIV (A0038 A23) (A0037 A23)))
  IF (LIST A23) & (ORDER NIL A23))
 ((FOLLOW (FIND NIL (A0038 A23)))
  IF (LIST A23) & (ORDER NIL A23)))
```

Retrieve the ith element, 1<=I<=N.

```
(((LE 1 I)
  IF (ELEMENT A22) & (IDENTIFIED-BY NIL A22 I))
 ((LE I N)
  IF (ELEMENT A22) & (IDENTIFIED-BY NIL A22 I))
 ((FOLLOW (RETRIEVE-FROM NIL A22 NIL))
  IF (ELEMENT A22) & (IDENTIFIED-BY NIL A22 I)))
```

Store a new value into the ith position, 1<=I<=N.

```
(((LE 1 I)
  IF (POSITION A33) & (IDENTIFIED-BY NIL A33 I)
     & (VALUE A15) & (NEW A15))
 ((LE I N)
  IF (POSITION A33) & (IDENTIFIED-BY NIL A33 I)
     & (VALUE A15) & (NEW A15))
 ((FOLLOW (STORE NIL A15 (INTO A33)))
  IF (POSITION A33) & (IDENTIFIED-BY NIL A33 I)
     & (VALUE A15) & (NEW A15)))
```

Insert a new element at position I, 1<=I<=N+1 causing elements numbered I,I+1,...,N to become numbered I+1,I+2,...,N+1.

```
(((POSITION (A0062 A18))
 IF (ELEMENT A18) & (NEW A18))
((IDENTIFIED-BY NIL (A0062 A18) I)
 IF (ELEMENT A18) & (NEW A18))
((LE 1 I) IF (ELEMENT A18) & (NEW A18))
((LE I (PLUS N 1)) IF (ELEMENT A18) & (NEW A18))
((FOLLOW (INSERT NIL A18 NIL (AT (A0062 A18))))
 IF (ELEMENT A18) & (NEW A18))
((ITEM-OF (A0063 A54 A18)
   NIL
   (SEQUENCE (PLUS I 1) (PLUS I 2)
      ELLIPSIS (PLUS N 1)))
 IF (ELEMENT A54)
    & (ITEM-OF A62 NIL
        (SEQUENCE I (PLUS I 1) ELLIPSIS N))
    & (IDENTIFIED-BY NIL A54 A62)
    & (NUMBER A62) & (ELEMENT A18) & (NEW A18))
((CAUSE (INSERT NIL A18 NIL (AT (A0062 A18)))
    (COME-ABOUT
       (AND (IDENTIFIED-BY NIL A54 (A0063 A54 A18))
            (NUMBER (A0063 A54 A18)))))
 IF (ELEMENT A54)
    & (ITEM-OF A62 NIL
        (SEQUENCE I (PLUS I 1) ELLIPSIS N))
    & (IDENTIFIED-BY NIL A54 A62)
    & (NUMBER A62) & (ELEMENT A18) & (NEW A18)))
```

REFERENCES

[1] Robert Balzer, Neil Goldman, and David Wile.
Informality in Program Specification.
IEEE Transactions on Software Engineering SE-4(2), March, 1978.

[2] Alan W. Biermann and Bruce W. Ballard.
Toward Natural Language Computation.
American Journal of Computational Linguistics 6(2), 1980.

[3] R.J. Bobrow.
The RUS System.
In B.L. Webber, R. Bobrow (editors), *Research in Natural Language Understanding*, . Bolt, Beranek and Newman, Inc., Cambridge, MA, 1978.
BBN Technical Report 3878.

[4] Ronald Brachman.
A Structural Paradigm for Representing Knowledge.
Technical Report, Bolt, Beranek, and Newman, Inc., 1978.

[5] Daniel L. Chester.
HCPRVR: An Interpreter for Logic Programs.
In *Proceedings of the National Conference for Artificial Intelligence*, pages 93-95. American Asociation for Artificial Intelligence, Aug, 1980.

[6] Fred J. Damerau.
Operating Statistics for the Transformational Question Answering System.
American Journal of Computational Linguistics 7(1):30-42, 1981.

[7] Fernando Gomez.
Towards a Theory of Comprehension of Declarative Contexts.
In *Proceedings of the 20th Annual Meeting of the Association for Computational Linguisitcs*, pages 36-43. Association for Computational Linguistics, June, 1982.

[8] C. Green.
The Design of the PSI Program Synthesis System.
In *Second International Conference on Software Engineering*. IEEE Computer Society, Oct, 1976.

[9] Jerry R. Hobbs.
What the Nature of Natural Language Tells us about how to Make Natural-Langauge-Like Programming More Natural.
In *SIGPLAN Notices*, pages 85-93. SIGPLAN, 1977.

[10] Ellis Horowitz and Sartaj Sahni.
Fundamentals of Data Structures.
Computer Science Press, Woodland Hills, CA, 1976

[11] Jonathan J. King.
Intelligent Retrieval Planning.
In *Proceedings of the National Conference for Artificial Intelligence*, pages 243-245. American Asociation for Artificial Intelligence, Aug, 1980.

[12] William Mark.
Rule-Based Inference In Large Knowledge Bases.
In *Proceedings of the National Conference on Artificial Intelligence*. American Association for Artificial Intelligence, August, 1980.

[13] Marjorie Templeton and John Burger.
Problems in Natural Language Interface fo DBMS with Examples from EUFID.
In *Conference on Natural Language Processing*, pages 3-16. Association for Computational Linguistics, Feb, 1983.