Chapter 4

Discussion and Conclusion

In this research, we have developed a small grammar which can translate a subset of English into Horn Clauses, and experimented with it over the Blocks World domain. We have also developed a translator for the experimental domain which can accept several tasks in English, translate them into Horn Clauses by using the grammar, and interpret them as programs and data.

In this chapter, we will discuss some points which have not been covered yet, and then summarize the related works. In addition, with the conclusions drawn from this research, we will discuss the directions for further research.

4.1 Further Discussion

In the examples of the last chapter, we could see that some of the descriptions of problem-solving methods were not very natural. There are some reasons for this unnaturalness. One of the reasons is the use of variables such as X and Y. Even though, by using variables, we can state our intentions clearly, readability is reduced, since variables do not represent any semantic information. If we use a noun phrase as a variable, e.g. "a block", it will increase the readability. However, this will cause some pronominal references and will increase the ambiguity. Another cause which makes some rules unnatural is the restricted types of formats for rules. Virtually, only one form of rule, i.e. if-then structure, is allowed. To represent some ideas naturally, we may need more structures like "repeat" and "do while". Also, to represent the complex problem-solving methods easily, we need some kinds of data structure,

e.g. array or record. In HCPRVR, the dominant data structure is *list* and, in the examples of the last chapter, we used it frequently.

In the first chapter, we observed the parallelism between Procedural Logic and English which eases the process of translation of English to Procedural Logic or vice versa. This research is about only one direction of the translation. But, the other direction seems to be feasible, too. In some senses, the translation from Horn clause programs to English seems to be more promising for automatic documentation of programs. We can produce an English description of a Horn Clause program with the reverse procedure of the translation discussed in this paper. One merit of this translation is that the grammar for the subset can be small. The system can explain the behavior of the program by using only a few combining roles for simple English sentences.

As for the problem of "consistency", we discussed the problem only when we asserted something. But, we have to consider the problem also when we delete something. It indicates that DB is non-monotonic. If we delete some facts (or rules) without checking, it may make DB inconsistent. For example, if we delete the fact "a red block is on the table" from DB of Figure 4-1, the resulting DB will be inconsistent. Furthermore, we have the fact "a blue block is on the red block," which means that we have a block which is on a non-existent block.

```
A red block is on the table.
A blue block is on the red block.

((INST 1 BLOCK COLOR RED) IF)
((INST 2 TABLE DET THE) IF)
((INST 3 BLOCK COLOR BLUE) IF)
((ON* (INST 1) (INST 2)) IF)
((ON* (INST 3) (INST 1)) IF)
```

Figure 4-1: Deletion from DB

A solution to this problem would be to resort on the consistency rules

again. Rules can specify the conditions which must be met in order to delete something. In the example above, the following consistency rule can be used to prevent the inconsistency shown above.

It is consistent to delete the fact that X is on Y
if X is clear.

More general solutions to the problem of consistency of the non-monotonic DB are discussed in [Doyle 79] [McCarthy 80] [Petrie 84],and the rule-based control of DB state change is discussed in [Nicholas 78b].

4.2 Related Works

This work deals with the translation of English program descriptions into Horn Clause programs by using a small grammar, and can be categorized as *Natural Language Computation*(NLC) or *Natural Language Processing*(NLP).

There have been lots of pros and cons about programming in natural language, and it is still controversial. A summary of arguments can be found in [Petrick 76]. The claims of proponents are that many potential users do not want to learn and use a formal language, and that, in some applications, natural language is a better tool for communication with the computer than formal languages. Also they claim that the current state of art in NLC is sufficiently advanced and many remaining problems can be solved. But, opponents argue that natural language is so ambiguous and vague that NLC system can make errors because of misunderstanding the input in natural language. They also argue that it is very hard to provide a large subset of natural language for computation in the near future. Recently, [Rosenschein 83] has stressed the importance of studying NLC in current environment. That is, we have to study NLC, because NLC is getting practical utility in some domains, e.g. natural data base query language, and the study interlinks related fields such as linguistics, philosophy, and psychology. In A.I., the study of NLC accelerates the study of "explicit models of mind", which is one of this field's ultimate goals, and contributes to progress in knowledge representation and acquisition.

[Biermann 76] compared the efforts and time needed to make programs using a subset of English and a formal language(PL/C). In this work, experimentation was done by asking a group of students to program in both languages about problems of arrays. The result from the statistics of time measurements and also from interviewing students showed that natural language fared better than PL/C.

Among the reports on individual projects related with this work, [Van Baalen 83] developed a system which constructs a recursive LISP program out of English sentences. This system accepts declarative sentences which are programs and imperative sentences which are function calls. The declarative sentence uses the "be" verb, and the basic translation mechanism is rather syntactic. For example, a sentence of the following structure

noun + of + NP1 + qualifier + is + NP2

will be translated into the following LISP function.

```
(noun (LAMBDA (X)
       (COND ((NP1* X)
              (COND ((qualifier* X) NP2*))))))

where NP1* is a predicate function to check if X is NP1,
      qualifier* is predicate function to check if X
      has the required qualification,
      and NP2* is a function which returns the value
      which is specified by NP2.
```

The system also has a list of words and their corresponding LISP functions(user-defined and system-defined), and uses it to choose the right function for each word used in input.

About translating English to Procedural Logic, there have been two previous works. [Wang 83] proposed the original idea of this kind of translation, and experimented with the idea on a domain of family hierarchy with a small grammar. [Simmons 84b] extended the idea to the extent that the grammar understands comparative statements. Their works mainly dealt with facts and rules which can make inferences from the given facts.

4.3 Conclusion

In chapter 1, it was argued that Procedural Logic is not only an excellent programming language, but also a good symbolism for representing some human problem solving. In addition, by showing parallelism between Procedural Logic and a small subset of English, we suggested that the mechanical translation from English program descriptions to Horn Clause programs was worth study.

In the next chapter, we introduced a small grammar which can translate a subset of English sentences into Horn Clauses. The translation is mainly based on the syntactic structures of simple sentences from which sentences of complex structures can be built. Also, to make it easy to identify the references to several objects, noun phrases were stored as frames. This made changes in some literals. A program which is based on Relaxed Unification algorithm was introduced to compare frames effectively.

In Chapter 3, an interpreter was described which uses the grammar discussed in Chapter 2, and is able to accept English definitions of Blocks World tasks. We experimented with the interpreter on an example domain, Blocks World, to show its power. Also, in this chapter, we discussed the mechanism which maintains DB consistency by using Consistency Rules. Throughout many examples on the domain, we showed how several tasks such as programming, querying, describing the Blocks World, and commanding the robot could be done in English in a unified way.

As mentioned in the first chapter, this research was one of a series of experiments relating to the automatic translation between English and Horn Clauses. This research, even though limited to a small example domain, has provided the following conclusions.

- Procedural Logic provides a unified system to accomplish a wide variety of computations, and, because procedural descriptions in English can closely parallel Procedural Logic descriptions, translations between them are facilitated.
- The grammar used in the translation can be made small because the

translation procedure does not need any domain-specific or semantic knowledge. The semantics of each predicate name and domain-specific knowledge should be defined and given by users in the form of rules and assertions whenever they are needed.

4.4 Further Research

Because we experimented with translation only in a simple domain, we need more experiments on other domains to get insights into the problems in this kind of translation. And, in further experiments on other domains, the following two points should be considered seriously.

- The characteristics of English descriptions of problem-solving methods should be studied to provide a subset of English which is small but which offers a reasonable and comfortable way of describing the problem-solving methods.
- The practical domains, in which this kind of translation is helpful, should be sought. One potential practical domain is expert systems such as Emycin. It has been already known that Procedural Logic can provide a good tool for implementing expert systems [Clark 80], because many expert systems use rules as the main structure of domain knowledge, and Procedural Logic can provide several facilities which an expert system has to have. The remaining problem is to determine if it is possible to convert English rules of an expert system into Horn Clauses by the translation procedure described in this thesis.

It is too early to evaluate the usefulness of this research. The accurate evaluation will be possible only after we get more experiences from further research.

Appendix A.

HCPRVR Program for Simple Sentences

```
(AXIOMS
'(((SIMSNT X V R)
  < (SIMSNT1 X V R1) (OR* (EQ* R1 (AND . R)) (EQ* R1 R)))))

(AXIOMS
'(((SIMSNT1 (X THERE . Y) V R)
   <
   (VBE X)
   !
   (SIMSNT1 (THERE X . Y) V R))
  ((SIMSNT1 (THERE X . Y) (_PREP U V) R)
   <
   (VBE X)
   (NP Y _NF U (Z . R1))
   (PREP Z)
   !
   (NP R1 _NF2 V R)
   (PREPARC NF Z NF2 _PREP))
  ((SIMSNT1 (THERE X . Y) (THERE-IS V) R)
   <
   (VBE X)
   !
   (NP* Y _NF V R))
  ((SIMSNT1 (IS IT X . Y) (_PRED V) R)
   <
   ! (SIMSNT1 (IT IS X . Y) (_PRED V) R))
  ((SIMSNT1 (IT IS X . Y) (_PRED V) R)
   <
   (OR* (ADJ X X1 _IDIOM _PRED)
        (AND* (PASTP X Z) (VERB Z Z1 Z2 _PRED)))
   ! (NP Y CLAUSES V R))
  ((SIMSNT1 (X . Y) V R1)
   < (VERB X X1 X2 X3) ! (VP NIL NIL (X . Y) V R1))
  ((SIMSNT1 (X . Y) V R1)
   <
```

```
   (VBE X)
   (MOOD QUERY)
   !
   (NP Y _NF V1 R)
   (VP V1 _NF (X . R) V R1))
  ((SIMSNT1 X V R1) < (NP X _NF V1 R) (VP V1 _NF R V R1))))

(AXIOMS
'(((NP (LIST X . R) LIST X R) < !)
  ((NP (THE X THAT . Y) CLAUSES V R)
   <
   (NOUN X _NF X1)
   !
   (SIMCONJSNT Y V2 R)
   (NORM V2 V))
  ((NP (X THAT . Y) CLAUSES V R)
   <
   (OR* (NOUN X _NF X1) (PRON X X1))
   !
   (SIMCONJSNT Y V2 R)
   (NORM V2 V))
  ((NP (X . Y) PRON* X1 Y) < (PRON X X1))
  ((NP (X . Y) VAR* X Y) < (VAR* X) !)
  ((NP (THAT . X) CLAUSES V R)
   < ! (SIMCONJSNT X V1 R) (NORM V1 V))
  ((NP (TO . X) CLAUSES V R) < ! (VP NIL NIL X V R))
  ((NP (X . Y) _NF (U DET X . V) R)
   < (ART X) ! (NP1 Y _NF (U . V) R))
  ((NP X _NF V R) < (NP1 X _NF V R))))

(AXIOMS
'(((NP* (AND . X) _NF (U . V) R)
   < (NP X _NF U R1) (NP* R1 _NF V R))
  ((NP* NIL _NF NIL NIL) < !)
  ((NP* (AND . X) _NF NIL (AND . X)) < !)
  ((NP* X _NF Y R)
   <
   (NP X _NF U (AND . R1))
   (NP* (AND . R1) _NF V R)
   (NORM (U . V) Y))
  ((NP* X _NF V R) < (NP X _NF V R))))

(AXIOMS
'(((NP1 (X . Y) _NF (U S X . V) R)
   <
   (ADJ X _FA _IDIOM _PRED)
   (NP1 Y _NF (U . V) R)
   (FRAMEARC _FA _NF S))
  ((NP1 (X . Y) _NF (U S X . V) R)
```

```
   <
   (NOUN X _NF1 _PRED)
   (NP1 Y _NF2 (U . V) R)
   (FRAMEARC X U S))
  ((NP1 (X Y . Z) _NF1 (X _ARC V) R)
   <
   (NOUN X _NF1 _PRED)
   (PREP Y)
   (NP Z _NF2 V R)
   (PREPARC _NF1 Y _NF2 _ARC))
  ((NP1 (X . Y) _NF (X) Y) < (NOUN X _NF W1))))

(AXIOMS
'(((VP _NP _NF (X NOT . Z) (UNLESS+ V) R)
   < (VBE X) ! (VP _NP _NF (X . Z) V R))
  ((VP _NP _NF (X NOT . Z) (UNLESS+ V) R)
   < ! (VP _NP _NF Z Y R))
  ((VP NIL NIL (X . Y) (_PRED U V) R)
   <
   (VERB X _VF _IDIOM _PRED)
   (NEQ* _IDIOM NIL)
   (NP Y _NF U R1)
   (REMB _IDIOM R1 R2)
   (NP R2 _NF2 V R))
  ((VP NIL NIL (X . Y) (_PRED V) R)
   <
   (VERB X _VF _IDIOM _PRED)
   (REMB _IDIOM Y R1)
   (NP R1 _NF V R))
  ((VP NIL NIL (X . Y) (_PRED V) R)
   <
   !
   (VERB X _VF _IDIOM _PRED)
   (NP Y _NF V R1)
   (REMB _IDIOM R1 R))
  ((VP _NP _NF (X Y . R) (_ARC _NP Y) R)
   <
   (VBE X)
   (ADJ Y _FA NIL _PRED)
   (FRAMEARC _FA _NF _ARC))
  ((VP _NP _NF (X Y . R) (_PRED _NP) R)
   < (VBE X) (ADJ Y _FA NIL _PRED))
  ((VP _NP _NF (X Y . Z) (_PRED _NP _NP2) R)
   <
   (VBE X)
   (ADJ Y _FA _IDIOM _PRED)
   (REMB _IDIOM Z R1)
   (NP R1 _NF2 _NP2 R))
  ((VP _NP _NF1 (X Y . Z) (_PRED _NP _NP2) R)
```

```
   <
   (VBE X)
   (PREP Y)
   !
   (NP Z _NF2 _NP2 R)
   (PREPARC _NF1 Y _NF2 _PRED))
  ((VP (V . V1) _NF1 (X . Y) (_ARC (V . V1) U) R)
   <
   (VBE X)
   (NP Y _NF2 (U . U1) R)
   (FRAMEARC V U _ARC))
  ((VP _NP _NF1 (X . Y) (_PRED _NP W) R)
   <
   (VBE X)
   (NP Y _NF2 V R)
   (REMDET V (U OF* W))
   (NOUN U X1 _PRED))
  ((VP _NP _NF1 (X . Y) (_PRED _NP) R)
   <
   (VBE X)
   (NP Y _NF2 (U . V) R)
   (NOUN U X1 _PRED))
  ((VP _NP _NF (X Y . Z) (_PRED _NP U) R)
   <
   (VBE X)
   (PASTP Y Y1)
   (VERB Y1 _VF _IDIOM _PRED)
   (REMB _IDIOM Z R1)
   (NP R1 _NF2 U R))
  ((VP _NP _NF (X Y . Z) (_PRED _NP) R)
   <
   (VBE X)
   (PASTP Y Y1)
   (VERB Y1 _VF _IDIOM _PRED)
   (REMB _IDIOM Z R))))

(AXIOMS
'(((OR*) < ! (FAIL))
  ((OR* X . Y) < X !)
  ((OR* X . Y) < (OR* . Y))))

(AXIOMS
'(((NORM (X) X) < !) ((NORM X X))))

(AXIOMS
'(((REMB NIL X X)) ((REMB (X . Y) (X . Z) R) < (REMB Y Z R))))

(AXIOMS
'(((NOUN LIST TYPE LIST*))
```

```
  ((NOUN BLOCK OBJ BLOCK*))
  ((NOUN CUBE OBJ CUBE*))
  ((NOUN MEMBER STATE MEMBER+))
  ((NOUN PYRAMID OBJ PYRAMID*))
  ((NOUN ROBOTARM OBJ ROBOTARM*))
  ((NOUN RULES TEXT NIL))
  ((NOUN TABLE OBJ TABLE*))
  ((NOUN FACT TEXT FACT*))
  ((NOUN INFERENCE TEXT INFERENCE*))))

(AXIOMS
'(((VERB ASSERT ASSERT NIL ASSERT*))
  ((VERB DELETE DELETE NIL DELETE))
  ((VERB DUMP PRINT NIL DUMP*))
  ((VERB MAKE CREATE NIL MAKE*))
  ((VERB PICK MOVE (UP) PICKUP))
  ((VERB PRINT PRINT NIL RPRINT))
  ((VERB PUT MOVE (DOWN) PUTDOWN))
  ((VERB PUT MOVE (ON) PUTON))
  ((VERB CHECK TEST NIL CHECK*))))

(AXIOMS
'(((VBE IS)) ((VBE ARE))))

(AXIOMS
'(((PRON NIL NIL)) ((PRON IT IT))))

(AXIOMS
'(((ADJ LARGER COMPAT (THAN) LARGER*))
  ((ADJ YELLOW COLOR NIL YELLOW*))
  ((ADJ BLUE COLOR NIL BLUE*))
  ((ADJ CLEAR PHYSTATE NIL CLEAR*))
  ((ADJ CONSISTENT STATE NIL CONSISTENT*))
  ((ADJ EQUAL RELATION (TO) EQ*))
  ((ADJ LARGE SIZE NIL LARGE*))
  ((ADJ RED COLOR NIL RED*))
  ((ADJ SMALL SIZE NIL SMALL*))
  ((ADJ WHITE COLOR NIL WHITE*))
  ((ADJ FALSE VALUE NIL FALSE*))
  ((ADJ SUPPORTABLE REL NIL SUPPORTABLE*))
  ((ADJ TRUE VALUE NIL TRUE*))))

(AXIOMS
'(((ART A)) ((ART AN)) ((ART ANOTHER)) ((ART THE))))

(AXIOMS
'(((PASTP ASSERTED ASSERT))
  ((PASTP DELETED DELETE))
  ((PASTP MADE MAKE))
```

```
  ((PASTP PICKED PICK))
  ((PASTP PUT PUT))
  ((PASTP CHECKED CHECK))
  ((PASTP PRINTED PRINT))))

(AXIOMS
'(((PREP ON)) ((PREP OF)) ((PREP ABOVE)) ((PREP UNDER))))

(AXIOMS
'(((FRAMEARC BLOCK PYRAMID SUB))
  ((FRAMEARC VAR* PYRAMID SUB))
  ((FRAMEARC COLOR X COLOR))
  ((FRAMEARC SIZE X SIZE))
  ((FRAMEARC PRON* PYRAMID SUB))))

(AXIOMS
'(((PREPARC X ON Y ON*))
  ((PREPARC X OF Y OF*))
  ((PREPARC X ABOVE Y ABOVE*))
  ((PREPARC X UNDER Y UNDER*))))
```

Appendix B.

HCTRANS

B.1 ELISP programs

```
(DE HCTRANS NIL
 (PROG (SENT SAVENUM)
       (SETQ QFLAG)
       (SETQ TFLAG)
       (SETQ LIMIT 3000.)
       (VARIABLES SOMETHING ANYTHING)
     A (SETQ VARLIST NIL)
       (SETQ SAVENUM OLDINSTNUM)
       (SETQ IVARNUM 0.)
       (TERPRI)
       (SETQ SENT (VARCHECK (READ-SENT)))
       (COND [(EQ MOOD '?)
              (ASSERT (MOOD QUERY) TEMP)
              (TRY (LIST 'QUERY SENT))
              (DELETE* (MOOD QUERY) TEMP)]
             [(RULE? SENT)
              (ASSERT (MOOD RULE) TEMP)
              (TRY (LIST 'RULE SENT))
              (DELETE* (MOOD RULE) TEMP)]
             [(COMMAND? SENT)
              (ASSERT (MOOD COMMAND) TEMP)
              (TRY (LIST 'COMMAND SENT))
              (DELETE* (MOOD COMMAND) TEMP)]
             [T (ASSERT (MOOD FACT) TEMP)
                (TRY (LIST 'FACT SENT))
                (DELETE* (MOOD FACT) TEMP)])
       (COND [(NULL VAL)
              (MSG T "(Processing of input aborted)" T)
              (CLEAR TEMP)
              (PUTPROP 'TEMP NIL 'AXIOMS)
              (SETQ OLDINSTNUM SAVENUM)])
       (TERPRI)
       (GO A)))
```

```
(DE READ-SENT NIL
 (PROG (SENT BUF TYPE)
       (TERPRI)
       (PROMPT 45.)
     A (SETQ BUF (READL))
       (COND [(AND [NOT (ATOM (CAR BUF))]
                   [NULL (CDR BUF)])
              (PRINT (EVAL (CAR BUF)))
              (TERPRI)
              (GO A)])
       (SETQ MOOD (CAR (LAST BUF)))
       (COND [(MEMQ MOOD (LIST '/. '?))
              (SETQ BUF (REVERSE (CDR (REVERSE BUF))))
              (PROMPT 42.)
              (RETURN (APPEND SENT BUF))]
             [(SETQ SENT (APPEND SENT BUF)) (GO A)])))

(DE VARCHECK (SNT)
 (COND [(NULL SNT) NIL]
       [(VAR SNT) (CAR (VARCHECK (LIST SNT)))]
       [(NOT (ATOM (CAR SNT)))
        (CONS (CONS (VARCHECK (CAAR SNT))
                    (VARCHECK (CDAR SNT)))
              (VARCHECK (CDR SNT)))]
       [(VAR (CAR SNT))
        (SETQ JJ (READLIST (APPEND '(V A R -)
                 (EXPLODE (CAR SNT)))))
        (AXIOMS (LIST (LIST (LIST 'VAR* JJ) 'LEXI)))
        (COND [(MEMBER JJ VARLIST) NIL]
              [T (SETQ VARLIST (CONS JJ VARLIST))])
        (CONS JJ (VARCHECK (CDR SNT)))]
       [T (CONS (CAR SNT) (VARCHECK (CDR SNT)))]))

(DE RULE? (SENT)
 (COND [(NOT (NULL VARLIST)) T]
       [(OR [MEMBER 'IF SENT] [MEMBER 'UNLESS SENT]) T]
       [(HEAD '(IT IS ALWAYS THE CASE) SENT) T]
       [(HEAD '(IT IS NOT THE CASE) SENT) T]
       [T NIL]))

(DE HEAD (X Y)
 (COND [(NULL X) T]
       [(EQ (CAR X) (CAR Y)) (HEAD (CDR X) (CDR Y))]
       [T NIL]))

(DE COMMAND? (SENT)
 (COND [(TRY (LIST 'VERB (CAR SENT) 'X 'Y 'Z)) T] [T NIL]))
```

```
{The following functions used in HCPRVR programs}

(DF VAR? (X) (OR [VAR (CAR X)] [HEAD '(V A R) (EXPLODE (CAR X))]))

(DE GETAVAR (V N)
 (PROG (VAR VAR1)
       (SETQ VAR1 (READLIST (CONS '_ (APPEND (EXPLODE V)
                                             (EXPLODE N)))))
       (PUTPROP VAR1 T 'VARIABLE)
       (SETQ VAR
        (READLIST (APPEND '(V A R - _) (APPEND (EXPLODE V)
                                               (EXPLODE N)))))
       (SETQ VARLIST (CONS VAR VARLIST))
       (RETURN VAR)))

(DF MAKEAVAR (X) (READLIST (CDDDDR (EXPLODE (CAR X)))))

(DE PAIRVARS (LST)
 (MAPCAR (F:L (XX) (CONS XX (READLIST (CDDDDR (EXPLODE XX)))))
                   LST))

(DF NOCONSRULE (X)
 (PROG (AXIOMS)
       (SETQ AXIOMS (GET 'CONSISTENT* 'AXIOMS))
       (SETQ X (CAR X))
    A (COND [(NULL AXIOMS) (RETURN T)]
            [(EQ (CAADR (CAAR AXIOMS)) X) (RETURN)])
       (SETQ AXIOMS (CDR AXIOMS))
       (GO A)))
```

B.2 HCPRVR programs

```
(AXIOMS
'(((ADDFRAME (INST N) _ARC Y)
   <
   (INST N X . X1)
   (DELETE (INST N X . X1))
   (ASSERT* (INST N X _ARC Y . X1)))))

(AXIOMS
'(((ALLTRUE* NIL) < !)
  ((ALLTRUE* (X . Y)) < X (ALLTRUE* Y) !)))

(AXIOMS
```

```
'(((AND*)) ((AND* X . Y) < X (AND* . Y))))

(AXIOMS
'(((APPEND* NIL X X))
  ((APPEND* (X . Y) Z (X . V1)) < (APPEND* Y Z V1))))

(AXIOMS
'(((ART A)) ((ART AN)) ((ART ANOTHER)) ((ART THE))))

(AXIOMS
'(((ASKS ((U . V) . W))
   < (ALLTRUE* ((U . V) . W)) (RPRINT* (YES (U . V))))
  ((ASKS X) < X (RPRINT* (YES X)) !)
  ((ASKS X) < (RPRINT (NO)))))

(AXIOMS
'(((ASS-TEMP)
   <
   (TEMP X)
   (DELETE* (TEMP X))
   (DELETE* X TEMP)
   (ASSERT* X)
   (FAIL))
  ((ASS-TEMP))))

(AXIOMS
'(((ASSERT* X) < (RPRINT* (Asserted: X)) (ASSERT X IF))))

(AXIOMS
'(((ASSERT-FACT (THERE-IS X)) < (ASS-TEMP) (REFERENCES X))
  ((ASSERT-FACT (X Y Z))
   < (FRAMEARC U V X) (REFERENCES (Z Y)) (ADDFRAME Y X Z))
  ((ASSERT-FACT X)
   <
   (ASS-TEMP)
   (REVERSE* X X1)
   (REFERENCES X1)
   (SETV* Y (SUBLIS (PAIRVARS VARLIST) 'X))
   (ASSERT* Y))))

(AXIOMS
'(((ASSERT-RULE X)
   <
   (RPRINT* (ASSERTED: X))
   (SETV* Y (SUBLIS (PAIRVARS VARLIST) 'X))
   (ASSERTB . Y)
   !)))

(AXIOMS
```

```
'(((CHANGEVAR (U DET U1 . U2)
        (RELMATCH (INST Z1 U . U2) (INST Z1)) NIL)
   <
   (MEMBER* U1 (A AN ANOTHER))
   !
   (NEWIVAR N)
   (SETV* Z1 (GETAVAR 'VAR N))
   (TEMP-ASSERT (INST Z1 U . U2)))
  ((CHANGEVAR (X . Y) V W) < (CHANGEVAR1 (X . Y) V W))
  ((CHANGEVAR X X NIL))))

(AXIOMS
'(((CHANGEVAR1 (U DET U1 . U2)
               (INST Z1)
               ((RELMATCH (INST Z1 U . U2) (INST Z1))))
   <
   (MEMBER* U1 (A AN ANOTHER))
   (NEWIVAR N)
   (SETV* Z1 (GETAVAR 'VAR N))
   (TEMP-ASSERT (INST Z1 U . U2)))
  ((CHANGEVAR1 (X . X1) Y Z) < (CHANGEVAR2 (X . X1) Y Z))
  ((CHANGEVAR1 X X NIL))))

(AXIOMS
'(((CHANGEVAR2 NIL NIL NIL))
  ((CHANGEVAR2 (X . Y) (U . V) W)
   <
   (CHANGEVAR1 X U W1)
   (CHANGEVAR2 Y V W2)
   (APPEND* W1 W2 W))
  ((CHANGEVAR2 X X NIL))))

(AXIOMS
'(((CHECK* X) IF (MAKE* X))))

(AXIOMS
'(((CHK-OBJS (X . Y) U) < (NOUN X X1 X2) ! (CHKIN (X . Y) U))
  ((CHK-OBJS (X . Y) V) < ! (CHK-OBJS1 (X . Y) V))
  ((CHK-OBJS X Y) < (PRON X X1) ! (CHKIN X Y))
  ((CHK-OBJS X X))))

(AXIOMS
'(((CHK-OBJS1 NIL NIL) < !)
  ((CHK-OBJS1 (X . Y) (U . V)) < ! (CHK-OBJS X U) (CHK-OBJS1 Y V))
  ((CHK-OBJS1 X X))))

(AXIOMS
'(((CHKIN X (INST N))
   <
```

```
   (REVERSE* X ((R1 . R2) X1 . R))
   !
   (CHKIN (R1 . R2) W)
   (REVERSE* R X2)
   (REMDET X2 X3)
   (RELMATCH (INST N . X3) (INST N))
   (X1 (INST N) W))
  ((CHKIN X (INST N)) < (PRON X X1) (INST N . Y))
  ((CHKIN (W DET THE . R1) (INST N))
   < (RELMATCH (INST N W . R1) (INST N)))
  ((CHKIN (W DET THE . R1) (INST N))
   <
   (RELMATCH (INST N W . R1) (INST N))
   !
   (FAIL))
  ((CHKIN (W DET Y . R1) (INST N))
   <
   (MOOD FACT)
   (MEMBER* Y (A AN ANOTHER))
   (NEWINSTNUM N)
   !
   (TEMP-ASSERT (INST N W . R1)))
  ((CHKIN (W DET THE . R1) (INST N))
   <
   (OR* (MOOD FACT) (MOOD RULE))
   (NEWINSTNUM N)
   !
   (TEMP-ASSERT (INST N W DET THE . R1)))
  ((CHKIN (X . X1) (INST N))
   <
   (UNLESS* (MEMBER* DET (X . X1)))
   (OR* (MOOD FACT) (MOOD RULE))
   (NEWINSTNUM N)
   !
   (TEMP-ASSERT (INST N X DET NIL . X1)))
  ((CHKIN (X DET Y . R) (X DET Y . R))
   <
   (OR* (MOOD RULE) (MOOD QUERY))
   (MEMBER* Y (A AN ANOTHER)))
  ((CHKIN X NIL)
   < (RPRINT (Uidentifiable Object: X)) ! (FAIL))))

(AXIOMS
'(((CLEAR-TEMP)
   < (TEMP X) (DELETE* X TEMP) (DELETE* (TEMP X)) (FAIL))
  ((CLEAR-TEMP))))

(AXIOMS
'(((COMMAND X) < (SNT X (Y IF)) (MULTI-CMDS Y))))
```

```
(AXIOMS
'(((CONJSNT (THEN . Y) NIL Y))
  ((CONJSNT (AND IF . Y) V2 R1)
   <
   !
   (SIMCONJSNT Y V R)
   (CONJSNT R V1 R1)
   (APPEND* V V1 V2))
  ((CONJSNT (AND UNLESS . Y) ((UNLESS+ . V) . V1) R1)
   <
   (SIMCONJSNT Y V R)
   (CONJSNT R V1 R1))
  ((CONJSNT X NIL X))))

(AXIOMS
'(((CONSISTENT (U . V)) < (NOCONSRULE U) !)
  ((CONSISTENT X) < (CONSISTENT* X))))

(AXIOMS
'(((DELETE X) < X (RPRINT* (Deleted: X)) (DELETE* X IF))))

(AXIOMS
'(((EQ* X X))))

(AXIOMS
'(((EXECUTE X) < X (REVERSE* X Y) (REFERENCES Y))))

(AXIOMS
'(((FACT X) < (SNT X (Y IF)) (MULTI-FACTS Y))))

(AXIOMS
'(((LIST* ((X . Y) . Z)))))

(AXIOMS
'(((MEMBER* X (X . Y)) < !)
  ((MEMBER* X (Y . Z)) < (MEMBER* X Z))))

(AXIOMS
'(((MEMPAIR (X Y) (X Y . Z)))
  ((MEMPAIR (X Y) (U V . Z)) < (MEMPAIR (X Y) Z))))

(AXIOMS
'(((MULTI-CMDS NIL) < !)
  ((MULTI-CMDS ((U . V) . Y))
   < ! (SING-CMD (U . V)) ! (MULTI-CMD Y))
  ((MULTI-CMDS X) < (SING-CMD X))))

(AXIOMS
```

```
'(((MULTI-FACTS NIL) < !)
  ((MULTI-FACTS ((U . V) . Y))
   < ! (SING-FACT (U . V)) ! (MULTI-FACTS Y))
  ((MULTI-FACTS X) < (SING-FACT X))))

(AXIOMS
'(((NEQ* X Y) < (UNLESS* (EQ* X Y)))))

(AXIOMS
'(((NEWINSTNUM N)
  < (SETV X (ADD1 OLDINSTNUM))
    (SETQ OLDINSTNUM X) (EQ* N X) !)))

(AXIOMS
'(((NEWIVAR N)
  <
  (SETV X (ADD1 IVARNUM)) (SETQ IVARNUM X) (EQ* N X) !)))

(AXIOMS
'(((OR*) < ! (FAIL))
  ((OR* X . Y) < X !) ((OR* X . Y) < (OR* . Y))))

(AXIOMS
'(((POINTERS X Y) < (VAR? X) (UNVAR X Y))
  ((POINTERS (INST N) (INST Y)) < (VAR? N) ! (UNVAR N Y))
  ((POINTERS (INST N) (INST N . X)) < (INST N . X) !)
  ((POINTERS (X . Y) (U . V))
   < (POINTERS X U) (POINTERS Y V) !)
  ((POINTERS X X) < !)))

(AXIOMS
'(((QUERY X)
   <
   (SNT X (Y IF))
   (TRANS-OBJS Y Z)
   (SETV* W (SUBLIS (PAIRVARS VARLIST) 'Z))
   (ASKS W))))

(AXIOMS
'(((REFERENCES (INST N))
   <
   (INST N . X)
   (DELETE* (INST N . X) IF)
   (ASSERT (INST N . X) IF))
  ((REFERENCES (X . Y))
   < ! (REFERENCES X) (REFERENCES Y))
  ((REFERENCES X))))

(AXIOMS
```

```
'(((RELMATCH (INST . X) (INST . X)))
  ((RELMATCH (INST N X . X1) (INST N X . Y))
   <
   (RELUNIF (INST N X . X1) (INST N X . Y)))
  ((RELMATCH (INST N) (INST N X . Y))
   <
   (INST N . Z)
   (RELMATCH (INST N . Z) (INST N X . Y)))
  ((RELMATCH (INST N X . Y) (INST N))
   <
   (INST N . Z)
   (RELMATCH (INST N X . Y) (INST N . Z)))
  ((RELMATCH (INST N X . X1) (INST N Y . Y1))
   <
   (NOUN X _NF1 U1)
   (NOUN Y _NF2 U2)
   (FRAMEARC X Y SUB)
   (RELUNIF (INST N Y . X1) (INST N Y . Y1)))
  ((RELMATCH (INST N Y . X1) (INST N X . Z))
   <
   (NOUN X _NF1 U1)
   (NOUN Y _NF2 U2)
   (FRAMEARC X Y S)
   (RELUNIF (INST N X S Y . X1) (INST N X . Z)))))

(AXIOMS
'(((RELUNIF (INST N X) (INST N X . Y)))
  ((RELUNIF (INST N X U V . R) (INST N X . Y))
   <
   (MEMPAIR (U V) Y)
   (RELUNIF (INST N X . R) (INST N X . Y)))))

(AXIOMS
'(((REMDET (X DET Y . R1) (X . R1))) ((REMDET X X))))

(AXIOMS
'(((REVERSE* NIL NIL))
  ((REVERSE* (X . Y) V)
   < (REVERSE* Y Y1) (APPEND* Y1 (X) V))))

(AXIOMS
'(((RPRINT* X) < (POINTERS X Y) (RPRINT Y) ')))

(AXIOMS
'(((RREAD) X) < (SETV* X (READ)))))

(AXIOMS
'(((RULE X) < (SNT X Y) (TRANS-OBJS Y Z) (ASSERT-RULE Z))))
```

```
(AXIOMS
'(((SETV* X Y) < (SETV Z Y) (EQ* X Z) !)))

(AXIOMS
'(((SIMCONJSNT NIL NIL NIL) < !)
  ((SIMCONJSNT (IF . X) NIL (AND IF . X)))
  ((SIMCONJSNT (UNLESS . X) NIL (AND UNLESS . X)))
  ((SIMCONJSNT (THEN . X) NIL (THEN . X)))
  ((SIMCONJSNT X (U . V) R)
   < (SIMSNT X U R1) (SIMCONJSNT R1 V R))
  ((SIMCONJSNT X NIL X))))

(AXIOMS
'(((SING-CMD X) < (CHK-OBJS X Y) (CONSISTENT Y) (EXECUTE Y))
  ((SING-CMD X)
   < (RPRINT* (Unacceptable Command : X)) ! (FAIL))))

(AXIOMS
'(((SING-FACT X)
   < (CHK-OBJS X Y) (CONSISTENT Y) (ASSERT-FACT Y))
  ((SING-FACT X)
   < (RPRINT* (Unacceptable Fact: X)) ! (FAIL))))

(AXIOMS
'(((SNT (IT IS ALWAYS THE CASE THAT . X) (V IF !))
   < ! (SIMSNT X V NIL))
  ((SNT (IT IS NOT THE CASE THAT . X) (V IF ! (FAIL)))
   <
   !
   (SIMSNT X V NIL))
  ((SNT (IF . X) (V1 IF . V))
   < ! (CONJSNT (AND IF . X) V R) (SIMSNT R V1 NIL))
  ((SNT (UNLESS . X) (V1 IF . V))
   <
   !
   (CONJSNT (AND UNLESS . X) V R)
   !
   (SIMSNT R V1 NIL))
  ((SNT X (U IF . V1))
   < (SIMCONJSNT X V R) (NORM V U) (CONJSNT R V1 NIL))))

(AXIOMS
'(((TEMP-ASSERT X) < (ASSERTB (TEMP X)) (ASSERTB X TEMP))))

(AXIOMS
'(((TRANS-OBJS X Y) < (MOOD RULE) (TRANSFORM X Y) (CLEAR-TEMP))
  ((TRANS-OBJS X Y)
   < (MOOD QUERY) (TRANSFORM1 X Y) (CLEAR-TEMP))))
```

```
(AXIOMS
'(((TRANSFORM (X Y . Z) (U Y . V))
   <
   (TRANSFORM1 X (U . R))
   (TRANSFORMN Z U1)
   (APPEND* R U1 V))))

(AXIOMS
'(((TRANSFORM1 (FAIL) ((FAIL))))
  ((TRANSFORM1 ! (!)))
  ((TRANSFORM1 (UNLESS+ . X) ((UNLESS+ . Y)))
   < ! (TRANSFORMN X Y))
  ((TRANSFORM1 (THERE-IS ((U . V) . Y)) Z)
   < (TRANSFORMN ((U . V) . Y) Z))
  ((TRANSFORM1 (THERE-IS X) Y) < (TRANSFORM1 X Y))
  ((TRANSFORM1 (X Y Z) ((RELMATCH (INST V1 V2 X Z) Y1)))
   <
   (FRAMEARC X1 X2 X)
   (NEWIVAR N)
   (SETV* V1 (GETAVAR 'VAR N))
   (NEWIVAR N1)
   (SETV* V2 (GETAVAR 'VAR N1))
   (CHK-OBJS Y Y1))
  ((TRANSFORM1 X (X2 . V))
   <
   (CHANGEVAR X X1 V)
   !
   (CHK-OBJS X1 X2)
   (REVERSE* X2 Y)
   !
   (REFERENCES Y))))

(AXIOMS
'(((TRANSFORMN NIL NIL))
  ((TRANSFORMN (X . Y) W)
   <
   (TRANSFORM1 X U) (TRANSFORMN Y V) (APPEND* U V W))))

(AXIOMS
'(((TRUE* NIL)) ((TRUE* X) < X)
  ((TRUE* ((X . X1) . Y)) < (X . X1) (TRUE* Y))))

(AXIOMS
'(((UNLESS* X) < X ! (FAIL)) ((UNLESS* X))))

(AXIOMS
'(((UNLESS+) < ! (FAIL))
  ((UNLESS+ X . Y) < X ! (UNLESS+ . Y))
  ((UNLESS+ X . Y) < !)))
```

```
(AXIOMS
'(((UNVAR U V) < (SETV* V (MAKEAVAR U)))))
```

Appendix C.

Planning

MAKE*

```
X is made if X is true.

The fact that X is on Y is made
        if it is made that X is clear and Y is clear
             and the robotarm is clear
    and if it is printed that X is put on Y
    and if X is put on Y.

The fact that the robotarm is clear is made
        if X is on the robotarm
    and if X is put down.

The fact that the X is clear is made
        if Y is on X
    and if it is made that Y is clear
           and the robotarm is clear
    and if it is printed that Y is put on the table
    and if Y is put on the table.

 X is made
        if X is equal to list (X1 . R)
    and if X1 is made
    and if R is made
    and if X is checked.

 ((MAKE* X) IF (TRUE* X))
 ((MAKE* (ON* X Y))
  IF
  (MAKE* ((CLEAR* X) (CLEAR* Y) (CLEAR* (INST 2 ROBOTARM))))
  (RPRINT* (PUTON X Y))
  (PUTON X Y))
 ((MAKE* (CLEAR* (INST 2 ROBOTARM)))
  IF
```

```
 (ON* X (INST 2 ROBOTARM))
 (RPRINT* (PUTDOWN X))
 (PUTDOWN X))
((MAKE* (CLEAR* X))
 IF
 (ON* Y X)
 (MAKE* (CLEAR* Y))
 (RPRINT* (PUTON Y (INST 1 TABLE)))
 (PUTON Y (INST 1 TABLE)))
((MAKE* X)
 IF
 (EQ* X (X1 . R)) (MAKE* X1) (MAKE* R) (CHECK* X))
```

CHECK*

X is checked if X is made.

```
((CHECK* X) IF (MAKE* X))
```

MEMBER+

It is always the case that X is a member of list (X . Y).

X is a member of list (Y . Z) if X is a member of Z.

```
((MEMBER+ X (X . Y)) IF !)
((MEMBER+ X (Y . Z)) IF (MEMBER+ X Z))
```

INFERENCE*

The fact that U is above V is an inference of X
if the fact that U is on W is a member of X
and the fact that W is on V is another member of X.

The fact that U is above V is an inference of X
if the fact that U is on W is a member of X
and the fact that W is above V is an inference of X.

```
((INFERENCE* (ABOVE* U V) X)
 IF
 (MEMBER+ (ON* U W) X) (MEMBER+ (ON* W V) X))
((INFERENCE* (ABOVE* U V) X)
 IF
 (MEMBER+ (ON* U W) X)
 (INFERENCE* (ABOVE* W V) X))
```

CONSISTENT*

It is consistent to make X

unless U is above U is an inference of X.

```
((CONSISTENT* (MAKE* X))
 IF
 (UNLESS+ (INFERENCE* (ABOVE* U U) X)))
```

Bibliography

[Biermann 76] Biermann, A. W.
Approaches to Automatic Programming.
In Morris Rubinoff and Marshall C. Yovits (editor), *Advances in Computer, vol 15*, . Academic Press, New York, 1976.

[Chester 80] Chester, D.
Using HCPRVR.
Technical Report, Department of Computer Science, University of Texas, 1980.

[Clark 78] Clark, K. L.
Negations as Failure.
In Gallaire, H., and Minker, J. (editor), *Logic and Data Bases*, . Plenum Press, New York, 1978.

[Clark 80] Clark, K. L., McCabe, F. G.
PROLOG: A Language for Implementing Expert Systems.
In Hayes, J. and Michie, D. J. (editor), *Machine Intelligence 10*, . Ellis and Horwood, 1980.

[Colmerauer 78] Colmeraruer, A.
Metamorphosis Grammars.
In Bolc, L. (editor), *Natural Language Communication with Computers*, . Springer-Verlag, New York, 1978.

[Doyle 79] Doyle, J.
A Truth Maintenance System.
Artificial Intelligence 12(3), 1979.

[Ehrlich 81] Ehrich, K.
Search and Inference Strategies in Pronoun Resolution: An Experimental Study .
In *Proceedings of the Conference: 19th Annual Meeting of the Association for Computational Linguistics.* Association for Computational Linguistics, 1981.

[Fikes 71] Fikes, R. E. and Nilsson, N. J.
Strips: A New Approach to the Application for Theorem Proving to Problem Solving.
Artificial Intelligence 2, 1971.

[Jaffer 83] Jaffer, J., Lassez, J., Lloyd, J.
Completeness of the Negation as Failure Rule.
In *Proceedings of the 8th International Conference on Artificial Intelligence.* Karlsruhe, West Germany, August, 1983.

[Kowalski 78] Kowalski, R.
Logic for Data Description.
In Gallaire, H., and Minker, J. (editor), *Logic and Data Bases*, . Plenum Press, New York, 1978.

[Kowalski 79] Kowalski, R.
Logics for Problem Solving.
North Holand, Limerick, Ireland, 1979.

[LeVine 80] LeVine, S. H.
Questioning English Text with Clausal Logic.
Master's thesis, University of Texas, December, 1980.

[McCarthy 80] McCarthy, J.
Circumscription - A Form of Non-Monotonic Reasoning.
Artificial Intelligence 13(1,2), 1980.

[Nicholas 78a] Nicholas, J. M. and Gallaire, H.
Data Base: Theory vs. Interpretation.
In Gallaire, H., and Minker, J. (editor), *Logic and Data Bases*, . Plenum Press, New York, 1978.

[Nicholas 78b] Nicholas, J. M. and Yazdanian, K.
Integrity Checking in Deductive Data Bases.
In Gallaire, H., and Minker, J. (editor), *Logic and Data Bases*, . Plenum Press, New York, 1978.

[Petrick 76] Petrick, S. R.
On Natural Language Based Computer Systems.
IBM Journal of Research Development , July, 1976.

[Petrie 84] Petrie, C.
The Problem of Consistency in Non-Monotonic Logics.
1984.
Submitted to AAAI-84.

[Rich 83] Rich, E.
Artificial Intelligence.
McGraw Hill, New York, 1983.

[Rosenschein 83] Rosenchein, S.
Natural Language Processing:Crucial for Computational Theories of Cognition.
In *Proceedings of the 8th International Conference on Artificial Intelligence.* Karlsruhe, West Germany, August, 1983.

[Simmons 83] Simmons, R. F.
A Text Knowledge Base for the AI Handbook.
Technical Report TR-83-24, Department of Computer Science, University of Texas, 1983.

[Simmons 84a] Simmons, R. F.
Computations from the English.
Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[Simmons 84b] Simmons, R. F.
Simple.Sim.
1984.
Lecture Material.

[Tseng 83] Tseng, S. H.
Questioning English Test with Clausal Logic: Case Studies of Three Texts.
Master's thesis, University of Texas, December, 1983.

[Van Baalen 83] Van Baalen, J. .
Using Natural Language to Solve Recursive Programming Problems .
Technical Report No. 2, Department of Computer Science, University of Wyoming, August, 1983.

[Wang 83] Wang, T. C.
Understanding a Subset of English.
1983.
Draft.

[Warren 77] Warren, D. H. D, Pereira, L. M., and Pereira, F.
PROLOG - The Language and its Implementation Compared with LISP.
SIGPLAN Notices 12(8), August, 1977.

[Wiederhold 84] Wiederhold, G.
Knowledge and Database Management.
Software, Computer Society, IEEE 1(1), February, 1984.

[Winograd 72] Winograd, T.
Understanding Natural Language.
Academic Press, New York, 1972.