COMPARE looks at each fact in the database in turn. If there is a match then the new set of values in VARS are added to ANS before setting VARS back to the empty list. COMPARE then continues working through the DATABASE to see if there are any other possible matches.

```
TO COMPARE :QUERY :DATA
  IF EMPTY? :DATA THEN STOP
  IF MATCH? :QUERY FIRST :DATA THEN MAKE
    "ANS FPUT :VARS :ANS
  MAKE "VARS []
  COMPARE :QUERY BUTFIRST :DATA
END
```

To see what MATCH? does, consider the case where the inputs are [OWNS ?SOMEONE AXE] and [OWNS JOSHUA AXE] in response to which MATCH? outputs TRUE and sets VARS to [?SOMEONE JOSHUA]. If the inputs are [OWNS ?SOMEONE AXE] and [KILLED ZACHARIAH AXE], then MATCH? outputs FALSE.

The real difficulties arise, however, if there is more than one variable involved. VALUE? is used to check if the variable has already been assigned a value for that fact in the database.

We have used here an alternative notation for conditionals in LOGO. TEST evaluates a condition. If the result is true then the actions following IFTRUE will be performed, otherwise the actions following IFFALSE will be carried out.

```
TO MATCH? :QUERY :FACT
  IF ALLOF EMPTY? :QUERY EMPTY? :FACT THEN
    OUTPUT "TRUE
  TEST FIRST FIRST :QUERY = "?
  IFTRUE IF NOT VALUE? FIRST :QUERY FIRST
    :FACT :VARS THEN OUTPUT "FALSE
  IFFALSE IF NOT ( FIRST :QUERY = FIRST :FACT )
    THEN  OUTPUT "FALSE
  OUTPUT MATCH? BUTFIRST :QUERY BUTFIRST
    :FACT
END
```

To see how VALUE? works, let's first consider the case where the inputs are ?IMPLEMENT, AXE, and [?MAN ZACHARIAH]. VALUE? tries to ascertain whether the variable ?IMPLEMENT could have the value AXE. There are three possibilities: ?IMPLEMENT already has a value, which is not AXE, and VALUE? outputs FALSE; ?IMPLEMENT already has the value AXE, and VALUE outputs TRUE; or ?IMPLEMENT does not have a value, so it is given the value AXE, and this information is added to VARS and TRUE is output.

```
TO VALUE? :NAME :VALUE :VLIST
  IF EMPTY? :VLIST THEN MAKE "VARS LPUT LIST
    :NAME :VALUE :VARS OUTPUT "TRUE
  TEST :NAME = FIRST FIRST :VLIST
  IFTRUE IF :VALUE = LAST FIRST :VLIST THEN
    OUTPUT "TRUE ELSE OUTPUT "FALSE
  OUTPUT VALUE? :NAME :VALUE BUTFIRST
    :VLIST
END
```

PRINTL simply arranges for the components of ANS to be printed out below each other.

```
TO PRINTL :LIST
  IF EMPTY? :LIST STOP
  PRINT FIRST :LIST
  PRINTL BUTFIRST :LIST
END
```

MORE COMPLEX ENQUIRIES

Our investigation will not go far, however, unless we can ask more complex questions, such as 'What implement killed Zachariah, and who owns such an implement?' In LOGO, this reads:

```
WHICH [[KILLED ZACHARIAH ?IMPLEMENT]
  [OWNS ?SUSPECT ?IMPLEMENT]]
```

WHICH now takes a list of queries as input and the values found will be those that make all of the queries true. If you then wish to ask a single query with this new form of WHICH the syntax we use is:

```
WHICH [[OWNS ?ANY KNIFE]]
```

We need make only minor alterations to these procedures:

```
TO WHICH :QUERIES
  FIND :QUERIES :DATABASE
  PRINT [NO (MORE) ANSWERS]
END
TO FIND :QUERIES :DATA
  MAKE "VARS []
  MAKE "ANS []
  COMPARE :QUERIES :DATA
  PRINTL :ANS
END
```

COMPARE now has a rather difficult job to do. Let's take [[KILLED ZACHARIAH ?IMPLEMENT][OWNS ?SUSPECT ?IMPLEMENT]] as an example input. COMPARE goes through the database, one fact at a time, to find a match for the first query, and ends up matching ?IMPLEMENT with AXE. The routine then considers the second query ([OWNS ?SUSPECT ?IMPLEMENT]), starting again from the beginning of the database. A match is found for the second condition, with the value of ?IMPLEMENT as AXE and ?SUSPECT as MATTHEW. There are no more queries, so this is a possible solution.

But we have not finished yet; there may be other values that satisfy the second query, while keeping ?IMPLEMENT as AXE. So COMPARE now proceeds through the database from the point it left off, and indeed finds a second solution with ?SUSPECT as JOSHUA. Of course, the procedure does not stop there, but continues searching the DATABASE. This time it reaches the end without finding any new matching values.

It is possible, however, that there is another solution to the first query — other than ?IMPLEMENT as AXE — so we must go back to the point where we found that match in the database and carry on from there. This process is called *backtracking*. In this case, there are in fact no other solutions.