Prolog: A Language for Teaching Computer-Based Business Problem Solving

David B. Paradice
Texas A&M University
College Station, Texas

Many business schools are currently redefining the content of their introductory computer course requirement. This change is being driven by a number of factors: increasing computer literacy on the part of incoming students, increasing availability of computer software packages that address specific needs, and increasing utilization of the computing resource to solve business problems to name a few.

Many curricula are moving away from BASIC or at least lessening the emphasis of BASIC as a vehicle for teaching computer-based, problem-solving techniques. BASIC is slowly being replaced by spreadsheet and file management software packages. This change has followed the current trend of applications software becoming more generally available and less expensive. Naturally, there is more emphasis on the utilization of such software.

A critical aspect of the learning process may be lost in the process of this transition. Specifically, the learning of problem-solving skills is being replaced by tool-use skills (Glasser 1986). This article looks beyond the current trend toward teaching application package tools and suggests that languages from the field of artificial intelligence may provide an attractive alternative to BASIC and application programs. One such language, Prolog (Programming in logic), is presented as a vehicle for introducing problem-solving processes using computer-based techniques.

The following sections briefly review some of the goals of the typical introductory computer course in a college of business, discuss how BASIC has been utilized in achieving these goals and why applications software is currently usurping the role of BASIC, and demonstrate how Prolog can be used to accomplish many of the tasks that are currently being used to justify the move from BASIC to spreadsheet and file management packages. Subsequently, some of Prolog's unique advantages over spreadsheet and file management packages are discussed. The final section presents a brief concluding remark.

The Purpose of the Introductory Computer Course

The DPMA (1981) guidelines for the introductory course specify the following course content:

1. Introduction to computer information systems (10%)
2. Processing concepts (10%)
3. Input/output (10%)
4. Memory storage (10%)
5. Data communication and distributed processing (10%)
6. Computer problem solving (10%)
7. Computer programming (30%)
8. Future of computers in society (10%)

Although obviously oriented toward the future data-processing professional, this course may be the only required computer skills course business students take if their major is not data processing or management information systems. The course description for this course will typically include phrases such as: ". . . introduction to the use of computers as a data processing and problem-solving tool for business." As indicated, most, if not all, introductory computer literacy

courses cover a great deal more than this (Barnes 1986). But for the students majoring in functional business areas other than data processing (or management information systems), the most important skill acquired in this course must be an ability to perceive how computers may be used as problem-solving tools.

Note that there are two goals to be achieved here. First is acquiring the knowledge required to use computers for data processing. Second is acquiring knowledge to apply the information required to use computers to solve problems. Unfortunately, the programming aspect of many introductory computer courses has frequently evolved as a means for addressing both goals (when, in fact, it is questionable whether learning to program should be a goal at all as observed by Tetenbaum and Mulkeen [1984]).

Undoubtedly, the relative importance of these two goals is considered equal. That is, acquiring the knowledge required to use computers for data processing is equally important as acquiring the knowledge required to use the computer to solve problems. This situation, however, is not reflected in the percentages listed, since problem solving receives only 10 percent of the allocated time and programming related activities receive at least 30 percent (and probably more). Clearly, nondata-processing majors taking this course will perceive little that appears relevant to their major.

Of course, many administrators would argue that in reality their introductory computer literacy course provides a more equitable treatment of programming and problem-solving topics. Let us assume that the actual course content reflects this position. If so, how do BASIC and the application software packages "measure up" in terms of providing a vehicle that allows a student to achieve both goals?

The Applicability of BASIC and Spreadsheets

Programming has long been used as a surrogate exercise for "thinking logically." As such, BASIC and other programming languages have been taught under the belief that "if a student can think through a problem in some fashion to the extent that he or she can write a program which reflects a sequence of steps which solves the problem, then the student has 'solved' the problem in a logical manner."

One look at a random sample of programming assignments from an introductory computer course will demonstrate how amusing this belief is. Although the steps to solve the problem may be sequential, in most cases these steps are far from logical. Frequently, the only problem adequately solved by the program is the problem of writing a program in the specified programming language (Watterman 1986). Of course, as more practice is gained, the programs typically become more logical and the student is able to shift his or her focus from the programming language to the problem to be solved. At the end of the semester, the better students will have achieved both goals.

Prolog has essentially one construct: the rule. Rules are specified in terms of goals to be satisfied.

A current trend is to integrate more applications packages into the introductory computer course. Although there is a great deal of importance in developing the skills necessary to use tools that will be utilized in a student's professional career, *certainly there is equal importance in teaching a student to recognize when a particular tool is applicable to a situation.* A shift from BASIC to a set of application software packages may not be addressing this issue. The student, now confronted with more "languages" to learn, is further distracted from focusing on the real problem to be solved. Before, the student was confronted with BASIC but upon achieving some level of competence was able to practice using BASIC to solve problems. Now, theoretically, the student learns a spreadsheet package, solves a spreadsheet problem, learns a database package, solves a database problem, etc. Are students really learning to use these packages to solve problems or are they solving the problem of using a particular software package?

It is in this light that a language such as Prolog may be seen as a viable alternative. Very simple Prolog programs may be constructed that solve problems perceived as very difficult by students at this stage of their college career. Since the programs are easily specified, students can focus on the problem to be solved, instead of the problem of writing the program.

Prolog as an Alternative

Prolog can be used to demonstrate how logic programming could be used to introduce business students to problem-solving approaches using computers. In the process, Prolog will be shown to provide a vehicle for learning to use the computer as a data processing tool *and* learning to solve problems.

Prolog has essentially one construct: the rule. Rules are specified in terms of goals to be satisfied. A rule that is always true may be called a fact. Rules may be conditional on the truth of other rules, in which case the rule is true when all of the "other" rules are true. The "other" rules represent "goals" because successfully proving their truth proves the truth of the conditional rule. So, a Prolog rule may be conceptually pictured as a set of goals to be achieved. When the conditions are satisfied (i.e., when the goals that represent the conditions are achieved), then the conclusion can be achieved.

Prolog can be used to demonstrate database-processing techniques. Suppose it is necessary to store cost data for 3 years. The following Prolog facts will suffice

```
cost (1986, 2000)
cost (1987, 2500)
cost (1988, 2290)
```

Notice how much more "logical" is Prolog's method for representing data. The data is stored in a way that is meaningful. Clearly, this is cost data for the years 1986, 1987, and 1988.

Database accessing is straight-forward. To determine the cost incurred in 1987, one simply specifies cost (1987, ____x), where ____x is a variable. Prolog handles all of the searching required to instantiate (or bind) the variable ____x to the value 2500.

How might Prolog be used in a business environment? Consider an accounting problem of constructing a balance sheet (a spreadsheet problem). Suppose that current assets are defined as the sum of cash, accounts receivable, and inventory. In Prolog, the rule for calculating current assets can be stated as follows:

```
calculate (current-assets ____year ____
  value) if
  calculate (cash ____year ____x) and
  calculate (account-receivable ____year
  ____y) and
  SUM (____x ____y ____intermediate-
  value) and
  calculate (inventory ____year ____z)
  and
  SUM (____intermediate-value ____z
  ____value).
```

Words that are entirely capitalized represent functions defined in Prolog (intrinsic functions). The rule can be read: The goal of calculating the value of current assets in a given year can be satisfied if the goal of calculating cash for the year supplies a value for x and the goal for calculating the value of accounts receivable for the year supplies a value for y and x and y may be added to get an intermediate value and the goal of calculating inventory for the year supplies a value for z and the intermediate value and z can be combined to produce the value (of current assets).

At first glance, this appears much more confusing than an equivalent BASIC statement. After all, what could be more straight-forward than 100 LET CURRENT ____ASSETS = CASH + AR + INVENTORY as specified in BASIC? However, BASIC requires that CASH, AR, and INVENTORY be calculated prior to the execution of this statement. This implies that calculations for these variables must logically occur prior to the execution of this statement and that usually the calculations (or subroutine invocations) must be physically in front of this statement.

Such preoccupation with location detracts from the problem-solving process involved. Indeed, the student can frequently tell you that values for CASH, AR, and INVENTORY are needed; otherwise how could CURRENT ____ASSETS be calculated?

Logic programming provides a unique means for teaching students to use computers to solve problems.

Their problem is in getting the computer to do the calculation.

Prolog, however, is "nearly non-procedural." (In some cases one must specify rules in a specific order to control Prolog's searching procedures. At this introductory level, however, this constraint can probably be safely ignored.) The rules for the calculations can be specified in any order. Consequently, the student is allowed to concentrate on the process of solving the problem. Data for a problem are represented as facts. The program is specified as a set of rules that manipulate the facts.

Once the rules have been established for creating the balance sheet, balancing is trivial. A single rule is necessary:

```
balance (____year) if
  calculate (total-assets ____year ____as-
  sets) and
  calculate (total-liabilities ____year
  ____liabilities) and
  SUM (____needed ____liabilities
  ____assets) and
  P (____needed ____year).
```

The translation is straight forward. The goal of balancing the balance sheet for a given year is accomplished if the calculation of total assets for the year produces a value (called ____assets) and the calculation of total liabilities for the year produces a value (____liabilities) and some amount needed when added to the amount of total liabilities is equal to total assets and the goal of printing (P) the amount needed is satisfied.

As the students become more proficient, new rules can be added that will automatically establish the financing alternatives for the amount needed to balance the balance sheet.

As a second example, consider a problem of calculating the net present value of a cost stream. A problem of this complexity would rarely be assigned in an introductory course (Trebby 1986), but the entire program can be specified in three sets of facts and rules:

```
cost (1986 2000)
cost (1987 2300)
cost (1988 2200)

savings (1986 1000)
savings (1987 2500)
savings (1988 3000)

discount (1986 1)
discount (1987 0.03)
discount (1988 0.03)

factor (1986 1)
factor (____year ____f) if
  discount (____year ____d) and
  SUM (1 ____d ____denominator) and
  TIMES (____reciprocal ____denomi-
  nator 1) and
  SUM (____next 1 ____year) and
  factor (____next ____nf) and
  TIMES (____reciprocal ____nf ____f)

pv (____year ____amount) if
  cost (____year ____c) and
  savings (____year ____s) and
  SUM (____x ____c ____s) and
  factor (____year ____f) and
  TIMES (____x ____f ____amount)

npv (1986 ____value) if
  pv (1986 ____value)
npv (____year ____value) if
  pv (____year ____v) and
  SUM (____next 1 ____year) and
  npv (____next ____nv) and
  SUM (____v ____nv ____value)
```

In this example, discount (a set of facts) specifies the cost of money in each year (1986 will be the base year for the calculations): factor is a rule that states that the present value factor is one in 1986 (otherwise the factor is the reciprocal of the discount rate in a given year times the factor of the previous year), pv merely states

that the present value is savings less the costs (facts) multiplied by the present value factor; and, npv states that the net present value of a value in 1986 is the same as the present value for that year (otherwise the net present value is the present value of this year's value plus the net present value of all other years' values).

Advantages of Logic Programming

In Prolog, as in other logic programming languages, a variable's scope is restricted to the rule in which it occurs. For example, in the rule for calculating current assets, the value of the variable ____year must be the same for all occurrences of the variable within the rule, but ____year may assume different values in other rules. Essentially, every rule is a subroutine. With Prolog, "subroutines" (and hence structured programming) are taught on the first day. Weeks may pass before these concepts are developed in BASIC. This structure means that additional rules may be entered later without regard to where the existing rules occur in the system.

Since the scope of Prolog variables is restricted to the rule in which the variable occurs and rules may be written in any order, each rule represents a complete specification. Therefore, the correctness of each rule may be determined separately from the occurrence of any other rule. This characteristic will allow instructors to point out flaws in the problem-solving process instead of the programming process. Once a rule has been correctly specified, the addition or deletion of any other rule to the system cannot alter the correctness of an existing rule.

Summary

The growth of artificial intelligence applications in general and logic programming in particular will surely impact the fundamental ways in which computer-based problem solving is taught. Logic programming is implemented in an essentially database environment where data are represented as facts and data are accessed through rules. Further, logic programming currently provides all of the tools required to implement spreadsheet type calculations. Consequently, all data-driven applications can be implemented in a logic programming language, although some extended features (e.g., report writing facilities) have yet to be implemented.

Additionally, logic programming provides a uniquely incremental, verifiable means for teaching students to use computers to solve problems. Consequently, instructors of introductory computer literacy courses would emphasize problem-solving techniques over programming techniques if a logic programming language as Prolog were used. Such an approach may be more beneficial to nondata processing majors in a college of business.

The intention of this article is not to suggest that Prolog should be the *only* computer-based technique taught, but rather that Prolog provides an attractive means for teaching problem-solving skills in a computer-based environment. Once the student has developed these skills, then he or she may begin to recognize where the special characteristics of specific application packages may be successfully and reasonably employed to solve new problems.

REFERENCES

Barnes, C. C. April 1986. Teaching computer literacy: A nontraditional approach. *Journal of Education for Business*, 311–14.

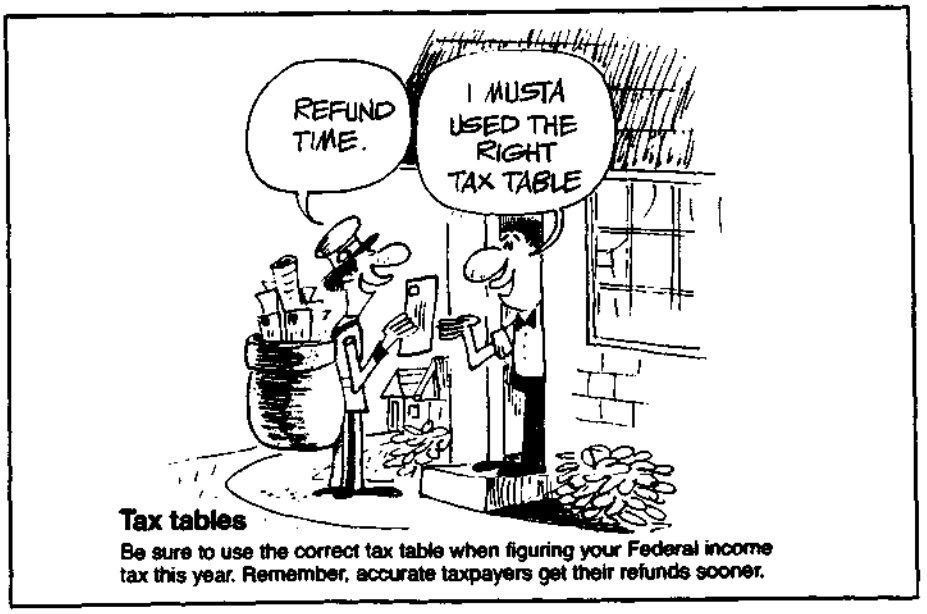
DPMA Education Foundation Committee on Curriculum Development. 1981. *DPMA model curriculum for undergraduate computer information systems education*, Park Ridge, IL: Data Processing Management Association Education Foundation.

Glasser, A. May 1986. On pounding with a hammer. *Collegiate Microcomputer*, 113–14.

Tetenbaum, T. J., and T. A. Mulkeen. November 1984. LOGO and the teaching of problem solving: A call for a moratorium. *Educational Technology*, 16–19.

Trebby, J. P. May 1986. Understanding the present and future value concept: A flowchart approach. *Journal of Education for Business*, 361–63.

Watterman, M. May 1986. Logic to code pedagogy in BASIC programming classes. *Journal of Education for Business*, 374–77.



Tax tables

Be sure to use the correct tax table when figuring your Federal income tax this year. Remember, accurate taxpayers get their refunds sooner.