

U092686

# PROLOG

## A Step Toward the Ultimate Computer Language

---

Ron Ferguson  
137 University Ave W, Apt 907  
Waterloo, Ontario N2L 3E6 Canada

---

What will the ultimate computer language be? What language will we be using once assembler language, BASIC, and Pascal have become museum pieces? Surprisingly, this question is easy to answer: there will be several ultimate computer languages. We even know what their names will be. They will be called English, Spanish, French, Russian, Chinese, etc. After all, the easiest language you could use to program a computer is the one you use to communicate with other people.

Unfortunately, programming a computer in English is still in the future. For a computer to understand English, it must be able to cope with the ambiguities inherent in any natural language. It must be able to deduce facts you don't bother specifying because they are "obvious." (Nothing is obvious to a computer unless it has been programmed to realize it is obvious. Everything must be stated explicitly and precisely.)

Today, though, we do have a language, called PROLOG, that simplifies the task of informing a computer about obvious (and not so obvious) facts. The name PROLOG is short for "PROgramming in LOGic"; however, you do not have to be familiar with formal logic theory to use PROLOG. In fact, the language is so simple a child can learn it. Yet its very simplicity makes it far more powerful than any other language currently available for use on microcomputers.

PROLOG is a programming language ideally suited to the manipulation of knowledge. A PROLOG program consists of facts about a certain subject. You can ask PROLOG questions and it will attempt to answer them

using the facts it has been told.

Facts are expressed in PROLOG in a concise manner. The fact that John is the father of Tom is expressed by the one-line PROLOG program:

```
father(john,tom).
```

The relationship, *father*, appears first, followed by the arguments (in parentheses) to which the relation applies. This data structure is called a *term*. The only potential area of confusion is the order in which the arguments are written. As a rule, the subject of the relation is the first argument and the object is the second. With this program, you can now ask PROLOG whether John is the father of Tom:

```
father(john,tom)?
```

to which PROLOG will respond:

```
yes
```

Note that the only difference between *telling* PROLOG that John is Tom's father and *asking* PROLOG whether John is Tom's father is the punctuation mark at the end of the statement. An assertion always ends with a period; a question always ends with a question mark. If you ask whether John is the father of Bill:

```
father(john,bill)?
```

# ELECTRIC SYSTEMS CORPORATION

Authorized Commodore service center  
 Repair of the complete line of Commodore products  
 In a hurry? Check our modular exchange program



HARDWARE:		SOFTWARE:	
CBM 8032 Computer, 80 Column	\$1195	OZZ	\$299
CBM 8050 Disk Drive	1395	Wordcraft 80	299
CBM 4032 Computer, 40 Column	995	Tax Preparation System	380
CBM 4040 Disk Drive	995	IRMA	380
CBM 4022 Printer	649	Dow Jones Portfolio Management System	115
CBM VIC 20 Computer	263	Personal Tax	55
CBM VS100 Cassette	68	Pascal	229
PET to IEEE Cable	33	Assembler Development Package	77
IEEE to IEEE Cable	39	Wordpro 4+	329
BASF Diskette, Box of 10	30		

**Order TOLL FREE  
 1+800-527-3135**

10 AM to 4 PM CDT Monday through Friday

Texas residents call 1+214-661-1370

VISA, MASTER CHARGE, MONEY ORDERS, AND C.O.D.  
 "Certified Check" accepted.

Units in stock shipped within 24 hours, F.O.B. Dallas, Texas.  
 All equipment shipped with manufacturer's warranty.

Residents of Texas, Louisiana, Oklahoma City and Tulsa,  
 Oklahoma must add applicable taxes.

Edlectic shortly will be announcing products that are designed to work with CBM systems.

1. ROMIO: two RS232 ports — three parallel ports — 26K EPROM memory-managed alternate character set, software controlled — EDOS (extended DOS).
2. Terminal program (options with ROMIO)
3. EPROM programmer
4. Front-end processor
5. Additional firmware to be announced

Be sure to write the address below for more information;  
 dealer inquiries welcome.

P.O. Box 1166 • 16260 Midway Road  
 Addison, Texas 75001 • (214) 661-1370

PROLOG will respond:

no

because it does not have any information to indicate that John is the father of Bill.

You can ask PROLOG more complicated questions such as who the father of Tom is by typing:

father(%who,tom)?

The % at the beginning of "%who" indicates that "%who" is a variable. When a question contains a variable, PROLOG attempts to assign a correct value to the variable. PROLOG will respond to the above question with:

%who = john

If PROLOG cannot find a correct value for the variable, it again responds no.

father(%who,bill)?

no

As is the case in most programming languages, it does not matter what name you use for a variable. You can have used "%x" instead of "%who". However, unlike most programming languages, a PROLOG variable's scope is limited to the statement in which it appears. If the same variable name is used in two separate statements, there is no connection between them. They are treated as different variables.

So far, PROLOG appears to be nothing more than an easy-to-use data-base language. You can store information and retrieve it later by asking questions. What makes PROLOG more than just another data-base language is that you can teach PROLOG how to manipulate the facts you have given it. Assume that you have made the following assertions:

father(bill,john).

father(john,tom).

From these assertions, you can deduce that Bill is the grandfather of Tom. PROLOG can also make this deduction if you tell it the fact that the father of the father is the grandfather. In PROLOG this fact is stated as a clause:

grandfather(%x,%z) — father(%x,%y),father(%y,%z).

— means "is implied by" or "is true if." The term to the left of — is true if the terms to the right of — are true. The term to the left is called a goal, and the terms to the right are called subgoals. The goal is true if the subgoals are true. The goal is also referred to as the head term of the statement.

Note that variables have been used in the clause to make a general statement about what it means for some-

one to be the grandfather of someone else. PROLOG can use this general statement to answer specific questions. If you ask the question:

```
grandfather(bill,tom)?
```

PROLOG will attempt to answer by setting "%x = bill" and "%z = tom" in the definition of grandfather. This creates an instance of the grandfather definition of the following form:

```
grandfather(bill,john) ←
father(bill,%y),father(%y,tom).
```

This states that Bill is the grandfather of Tom if Bill is the father of a person who is the father of Tom. By the first two assertions, PROLOG knows that Bill is the father of John and John is the father of Tom. Therefore, PROLOG will respond with a yes.

If you ask PROLOG to find two people such that one is the grandfather of the other:

```
grandfather(%x,%y)?
```

PROLOG will respond:

```
%x = bill, %y = tom
```

### Nonprocedural Languages

Most computer languages currently in use (such as BASIC and Pascal) are procedural: a computation is performed by executing a series of actions in a precise order. Each statement of a procedural language represents only one step in an algorithm. This means that the correctness of an individual statement cannot be determined by examining the statement by itself. Instead, you must examine the entire algorithm in which the statement occurs to determine if the statement is correct. For example,  $I = I + 2$  is a typical statement in a procedural language. If you are asked to determine whether the statement is correct, the most you can say is that its syntax is correct (ie: it is a valid statement in the language). This does not necessarily mean that the statement is the cor-

rect one to use at that particular stage of the algorithm. Perhaps the correct statement should be  $I = I * 2$ . You cannot tell without looking at the rest of the code in which the statement appears.

In PROLOG, on the other hand, you can determine whether a statement is true by examining that statement only. The correctness of the PROLOG statement that defines grandfather can be determined independently of the rest of the PROLOG program. A statement in a PROLOG program corresponds to an entire subroutine in a conventional programming language. Thus PROLOG programs are extremely modular. PROLOG carries the "divide-and-conquer" approach of structured programming one step further.

Another advantage of nonprocedural languages is that the order in which statements occur is irrelevant. Each PROLOG statement represents a fact, and it does not matter in what order PROLOG is told the facts. This means that you can increase the power of a program by adding new statements, and in most cases this does not require any modification of the statements that are already there. For example, the definition of grandfather, given earlier, is true, but it is only a partial definition. One must add the following statement to obtain a complete definition:

```
grandfather(%x,%y) ←
father(%x,%z),mother(%z,%y).
```

Assume that the following assertions are also made:

```
mother(jane,alice).
father(bill,jane).
```

Now if you ask for a grandchild of Bill:

```
grandfather(bill,%grandchild)?
```

PROLOG will respond:

```
%grandchild = tom
```

You can also find out if Bill has another grandchild by

- ANALOG I/O -
  - Low/High Level A/D
  - X-Ducer Compensation
  - 16 to 64 Channel MUX
  - Differential/Single End
  - Programmable Gain
  - 12/14 Bit Resolution
  - Video Digitizers
  - Multiple DACs
- DIGITAL I/O -
  - Optically Isolated
  - High Level AC or DC
  - Pulse Count I/O
  - Programmable Timers
  - Digital Multiplexers
  - UNI-BUS/IEEE Drivers
  - Voice Synthesizers
- COMMUNICATIONS -
  - Synchronous to 800KB
  - Asynchronous to 24KB
  - Serial RS-232 or 20 Ma
  - IEEE 488 - 78 GPIB
  - Distributed Systems
  - BISync - DEC net - X 25

Z80/Z8000

LSI-11/PDP-11

Data Acquisition

ASC Computers



8085/8086/88

6800/09/68000

Multi-Processing

#### ASC MICRO-COMPUTER SYSTEMS-

Featuring a full range of industry standard 8 and 16 Bit Micro-Processors, Memories, I/O Controllers, Analog/Digital Modules and Communications Units.

Expedited delivery on the latest technology in Micro-Computers, Disk Memories and Peripheral options, including Software and System Integration.

ASC Micro-Computer Systems are offered with optional IEEE/S-100 Bus, INTEL MULTI-Bus, MOTOROLA EXOR-Bus, or STD-Bus compatible Micro-Processors, Memories, I/O Controllers, and Enclosures to your specifications.

Call ASC for prompt quotations on Micro-Computer configurations.

29401 Harper Ave.-  
St. Clair Shores  
Michigan 48081

ASC Computer Systems 313  
779-8700

- PERIPHERALS -
  - Flexible Disks to 1Mb
  - Winchester Disks to 360Mb
  - Magnetic Tape Units
  - Printers/Terminals
  - Video Cameras
  - Color Graphics
  - CRT Displays/Plotters
- SYSTEMS -
  - Modular Expandability
  - Industrial Enclosures
  - Rack-Mounting Cabinets
  - I/O Termination Panels
  - Portable SAS Option
  - Control Console/Displays
  - Development Systems
- SOFTWARE -
  - CP/M Operating Systems
  - Multi-User Support
  - Fortran, Basic, Pascal
  - DAS - Process Languages
  - MACRO & Assembler Languages
  - Network Communications
  - Graphics and Applications

typing ? . Now PROLOG will respond:

```
%grandchild = alice
```

If you ask whether Bill has other grandchildren besides Tom and Alice by typing ? once more, PROLOG will respond with a no.

The two partial definitions for grandfather can be combined into one statement:

```
grandfather(%x,%y) - father(%x,%z),
(father(%z,%y);mother(%z,%y)).
```

The semicolon between "father(%z,%y)" and "mother(%z,%y)" means that one or the other must be true.

A term can have any number of arguments. To say that Alice is pretty, you would type:

```
pretty(alice).
```

Now you can ask PROLOG to find a pretty grandchild of Bill, as follows:

```
grandfather(bill,%grandchild),pretty(%grandchild)?
```

to which PROLOG will respond:

```
%grandchild = alice
```

## How PROLOG Works

To use PROLOG you do not need to know how it actually arrives at its answers. If you have specified all the required information, PROLOG will find the answer as if by magic. However, it is interesting to know how it goes about coming up with a solution.

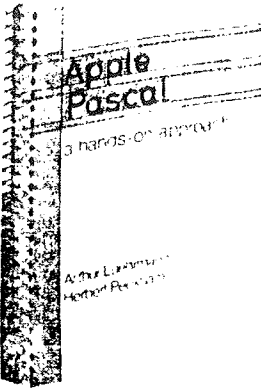
Basically PROLOG attempts to solve goals from left to right. For a given goal, PROLOG attempts to find a statement whose first term (the only term in an assertion; the term to the left of the - in a clause) can be made to match the goal. It then attempts to solve the subgoals of that statement. Of course, if the statement is an assertion, there are no subgoals. If the subgoals can be solved, PROLOG then proceeds to the next goal. If one of the subgoals cannot be solved, PROLOG backtracks and tries to find another statement whose head term matches the goal. If there are no untried statements left, PROLOG realizes it cannot solve this particular goal.

This does not necessarily mean there is no solution to your original question. If the goal PROLOG is working on is actually a subgoal of one of your original goals, there may be an alternate solution of the original goal that does not involve the failed subgoal. PROLOG will backtrack further and try to find an alternate solution; it gives up only when it can find no solution to any of your original goals.

Let us examine in more detail how PROLOG works.

**Listing 1:** Assertions that specify the environment. PROLOG attempts to answer questions based on facts and relations that it knows, stated in special syntax. Questions are broken into goals to be achieved from left to right.

```
father(bill, john).
father(john, tom).
father(bill, jane).
mother(jane, alice).
grandmother(%x,%y) <- father(%x,%y), (father(%z,%y);mother(%z,%y)).
pretty(alice).
```



**Apple Pascal**  
a hands-on approach  
by Arthur Luehrmann and Peckham

## McGraw-Hill Bookstore

Please print clearly.

**The Only Book That Is Machine Specific in Teaching Pascal on the Apple**

### APPLE-PASCAL

by Peckham and Luehrmann

- Teaches the total beginner to become competent in programming in Pascal
- Offers concrete experiences in creating, running and debugging actual programs in Pascal

This hands-on guide will teach you about the editor, the operating system commands, even the keyboard layout and labels of the Apple computer. 384 pp. \$13.95

McGraw-Hill Bookstore  
1221 Avenue of the Americas  
N.Y., N.Y. 10020

Please send me \_\_\_\_\_ copies of  
**APPLE-PASCAL** at \$13.95 each.

MasterCard \_\_\_\_\_ Visa \_\_\_\_\_ AmerExp \_\_\_\_\_

Account No. \_\_\_\_\_ Expires \_\_\_\_\_

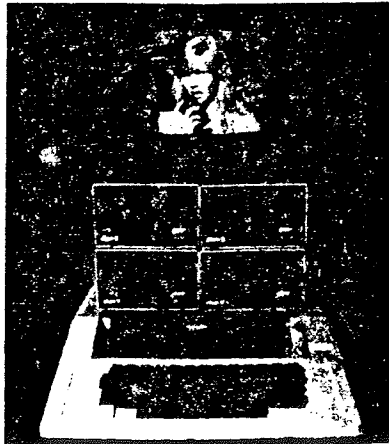
Name \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

Add sales tax plus \$2.50 postage, handling

**\*ATTENTION ALL APPLE USERS\***



**(602) 998-9411**

Desert Computer announces a new online software order service available 24 hours a day. Our complete catalog and order information are available on this system and on the Source. On the Source use mail to TCH860 to order and use >BASIC (11)TCH860>READ to view the catalog. Our system can be reached at the number above. If you do not have a modem you can send a sase to the address below or use the reader service number. We carry nationally known brands at discount prices.

**Take the BYTE out of Software Cost**  
 DESERT COMPUTER P.O. BOX 4841 SCOTTSDALE, AZ 85261

**Components Express, Inc.**  
*"Have you missed your computer lately?"*  
 1380 E. Edinger, Unit CC Santa Ana, CA 92705 (714) 558-3972

**BROAD BAND MICROWAVE RECEIVER SYSTEM**  
 1.8GHZ to 2.4 GHZ

only **\$295.00**

**With built-in-converter to channel 2, 3, or 4 of any standard TV set.**

**RANGE** Line of sight to 250 miles  
**SCOPE** Will receive within the frequency band from satellites, primary microwave stations, and repeater microwave booster stations  
**CONTENTS:** Packaged in 19"x19"x4 1/2" corrugated carton complete with:

- 24" Dish
- Feed-Horn Receiver
- Mounting Bracket
- Mounting Clamp
- Instructions
- 300 Ohm to 75 Ohm Adapter
- 750 Ohm to 300 Ohm Adapter
- 60 Feet Coax Cable with Connectors
- 3 Feet Coax Cable with Connectors

**WARRANTY**  
 180 days for all factory defects and electronic failures for normal useage and handling. Defective sub assemblies will be replaced with new or re-manufactured sub assembly on a 48 hour exchange guarantee.

This system is not a kit and requires no additional devices or equipment other than a TV set to place in operation. **DEALER INQUIRIES INVITED.**

Assume that the statements in listing 1 have been made. Now assume that PROLOG is asked this question:

grandfather (bill, %grandchild), pretty(%grandchild)?

Because PROLOG attempts to answer questions by solving goals one at a time from left to right, it will first attempt to solve "grandfather(bill, %grandchild)". It searches for a clause whose first term is grandfather. There is only one clause whose head term is grandfather, so PROLOG tries to match "grandfather (bill, %grandchild)" with "grandfather(%x, %y)". A match can be made by setting "%x = bill" and "%y = %grandchild". This substitution is applied to the subgoals of that statement, which may be expanded to:

father(bill, %z), (father(%z, %grandchild);  
 mother(%z, %grandchild))

PROLOG now attempts to solve "father(bill, %z)". The first three statements have father as a head term, so PROLOG tries them one after the other. The first statement matches if "%z = john"; it is an assertion and has no subgoals. Therefore, PROLOG can proceed to the next goal in the expanded statement. Here there is a choice between "father(john, %grandchild)" and "mother(john, %grandchild)". PROLOG attempts to solve the first alternative. It tries to match "father(john, %grandchild)" with the first given statement, but fails because the first arguments do not match. It then tries the second statement, and this time succeeds with the substitution "%grandchild = tom".

Now all the goals of the expanded statement have been satisfied, so PROLOG attempts to solve the last goal of the original question. Since "%grandchild = tom", the goal is "pretty(tom)". There is only one clause whose head term is "pretty", but "pretty(tom)" does not match "pretty (alice)". At this point, PROLOG backtracks to the last place where there was a choice—between "father(john, %grandchild)" and "mother(john, %grandchild)"—and selects the alternate choice: "(mother(john, %grandchild)". This choice does not work either because the fourth statement is the only one whose head is "mother" and "mother(john, %grandchild)" cannot match "mother(jane, alice)".

So PROLOG backtracks further and tries to solve the first goal of the expanded statement again.

First, it tries to match "father(bill, %z)" with the second statement but fails. Then it tries to match "father (bill, %z)" with the third statement, and this time succeeds by setting "%z = jane". Now PROLOG attempts to solve the second part of the expanded statement, which is a choice between "father(jane, %grandchild)" and "mother(jane, %grandchild)". Once again PROLOG attempts to solve "father(jane, %grandchild)" first but fails. Then it attempts to solve "mother(jane, %grandchild)". This time it succeeds by matching "mother (jane, %grandchild)" and setting "%grandchild = alice".

PROLOG now tries to solve the last goal in the original question, which is now "pretty(alice)". This goal matches the sixth statement, so "%grandchild = alice" is an answer to the original question.

### Controlling Robots

Now consider how a PROLOG program could be used to control a robot. Assume that a human and a robot are inside a rocket that has landed on the surface of a planet. The rocket has an airlock. On the planet there is a building that contains rocket fuel. There is also a cave with a key to the building. The cave also contains gold. The robot is able to lift the key, the fuel, or the gold (but it cannot lift the rocket). This situation can be described in PROLOG as shown in listing 2.

The statements in listing 2 represent the state of the robot's environment. As the robot interacts with the environment, some of the statements may cease to be true. To keep the description of the environment up to date you must have a way to delete statements no longer true. PROLOG provides a built-in function, called *delete*, to eliminate specified statements; another built-in function, called *assert*, can be used to add statements.

Now you can specify commands for the robot to attempt to obey. The first command will make the robot fetch an object to a specified place (see listing 3a).

The first subgoal, "inside(%object, %place)", checks to see if the object is already where you want it. If it is, the robot does not need to do anything. If it isn't, the robot must pick up the object, move to the required place and drop it. Note that the terms "pickup(%object)", "moveto(%place)", and "drop(%object)" have been grouped together within parentheses to show that the semicolon operator applies to all of them.

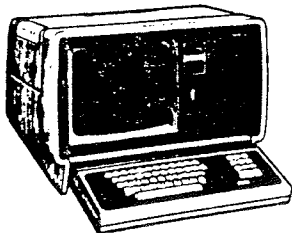
In order for the robot to pick up an object (see listing 3b), the object must be liftable. The robot must move to the place where the object is in order to pick it up. Assuming that the robot can carry only one thing at a time, it must be empty-handed when it picks up an object.

**Listing 2:** Hypothetical environment found in a simulated space expedition. Through a series of commands specified in listing 3, a robot can be made to perform complex tasks, such as exiting the craft, and finding and returning with fuel—all with a simple instruction from the user.

```
inside(human, rocket).
inside(robot, rocket).
inside(fuel, building).
inside(key, cave).
inside(gold, cave).
entrance(airlock, rocket).
entrance(door, building).
entrance(hole, cave).
closed(airlock).
closed(door).
liftable(key).
liftable(fuel).
liftable(gold).
```

## TEXAS COMPUTER SYSTEMS

# TRS-80 COMPUTERS



**Model II 64K \$3288**

An excellent computer for your business needs. Easy expandability & compatibility. No formal operator training needed. All accessories available—disk expansions, printers, software, at our low discount prices. Our fast, fully insured air freight service can assure most deliveries within seven days after payment is received.

**Model III 16K \$835** With TCS Memory:  
**Model III 32K \$979** Model III 32K \$909  
**Model III 48K \$1089** Model III 48K \$969

**Model III 48K 2 Disk RS232 \$2100**  
**Model III 32K 1 Disk \$1729**  
**Model III 48K 1 Disk \$1849**



### Corvus Hard Disks \$Call

5, 10, or 20 megabytes of storage for the Model I, II, or III, configured with TRSDOS, NEWDOS 80, or CP/M for one or several computers sharing a single hard drive simultaneously. Easy expandability and compatibility. Also fits Apple, Superbrain, Altos, and most other computers. Don't wait on other's promises. We can deliver this proven system now. Call us for the lowest price!

**Pocket Computer & Acc. \$Call**

Pocket Computer Printer Interface in stock

### Color Computer

**4K Level 1 \$319**  
**16K Level I \$439**  
**16K Extended Basic \$489**

**With TCS Memory:**

**16K Level I \$369**  
**16K Extended Basic \$449**

## Epson Printers \$Call

Letter quality matrix printer has full software control of 40, 80, 66 or 132 columns. 80 cps bidirectional tractor feed, disposable printhead. \$300 less than nearest competitive printer. Lists \$645. Call for our low price.

**MX-80 Tractor Feed** **MX-100 Graphtrax, Friction**  
**MX-80 FT Friction and** **and Tractor up to 15"**  
**Tractor** **wide.**  
**Graphtrax for MX-80, MX-80 FT, graphics option.**

### Word Processor Package \$2679

Includes 2 Disk Model III with 48K, Epson MX-80 Tractor Feed with cable, and word processing software ready to operate. Lists at \$3300. Our low price special this month: **\$2679**. For MX-80 FT Tractor and Friction, add \$99.

• Payment: Money Order, Cashier's Check, Certified  
 Check, Personal checks take 3 wks. VISA, MC  
 add 3%

• Prices subject to change any time  
 • No tax out-of-state. Texans add 5%.  
 • Delivery subject to availability  
 • Shipping extra, quoted by phone

## TEXAS COMPUTER SYSTEMS

**Box 951, Brady Texas 76825**

For fast, efficient service, we can air freight from Dallas to major a/p near you. Call for information.

**Toll Free Number 800-433-5184**

**Texas Residents 817-274-5625**

**Listing 3:** Fundamental commands built to control the hypothetical robot. In each case, the commands are constructed of goals and subgoals that are either basic enough for the robot to perform directly or are further broken into subgoals.

```
3a
  fetch(%object,%place)<- inside(%object,%place);
                          (pickup(%object),moveto(%place),drop(%object)).

3b
  pickup(%object)<- liftable(%object),
                   inside(%object,%place),moveto(%place),
                   emptyhanded,assert(holding(%object)).

3c
  emptyhanded <- (holding(%object),drop(%object));true.

3d
  drop(%object)<- delete(holding(%object)).

3e
  moveto(%place) <- inside(robot,%place);
                   (inside(robot,%place2),leave(%place2),enter(%place));
                   (outside(robot),enter(%place)).

3f
  leave(%place) <- entrance(%x,%place),((closed(%x),open(%x));true),
                   delete(inside(robot,%place),assert(outside(robot))),
                   ((holding(%object),delete(inside(%object,%place))),
                   assert(outside(%object)));true).

  enter(%place) <- entrance(%x,%place),((closed(%x),open(%x));true),
                   delete(outside(robot)),assert(inside(robot,%place)),
                   ((holding(%object),delete(outside(%object))),
                   assert(inside(%object,%place)));true).

3g
  open(door) <- (holding(key);(inside(key,%place),pickup(key),leave(%place))),
               delete(closed(door)).
```

Finally, you must assert that the robot is now holding the object.

In order to be *empty-handed*, the robot must drop whatever it is holding (see listing 3c). If it is not holding anything, it is already empty-handed, so the built-in true function (which always succeeds) is executed.

When the robot *drops* something, you must delete the statement that says it is holding the object (see listing 3d).

Depending on where the robot is located initially, there are three possible actions that the robot must perform to *move* to a specified place (see listing 3e).

- If it is already at the specified place, it does not need to do anything.
- If it is inside some other place, it must leave that place and enter the specified place.
- If it is already outside, it must enter the specified place.

To leave a place (see listing 3f), if the entrance is closed, the robot must open the entrance. You must also remember to delete the fact that the robot is inside the

place, and assert that the robot is now outside. If the robot is holding something, you must assert that the object also moves outside. Entering a place is accomplished similarly. To open the door to the building, the robot must either be holding the key or it must pick up the key (see listing 3g). Assume the robot can open the rocket's airlock automatically:

```
open(airlock — delete(closed(airlock)).
```

Now you can order the robot to fetch the gold to the rocket with the command:

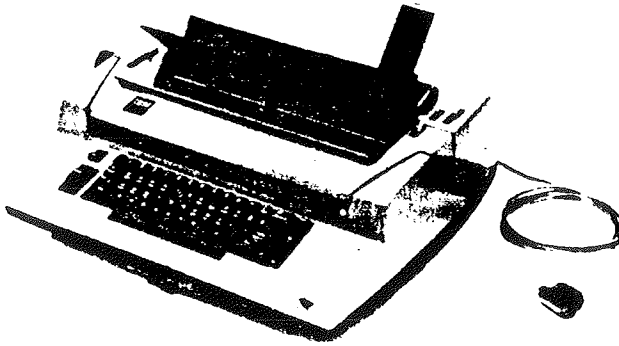
```
fetch(gold,rocket)?
```

The robot will leave the rocket, enter the cave, pick up the gold, and return to the rocket. If you ask the robot to fetch the fuel from the building by typing:

```
fetch(fuel,rocket)?
```

# In Less Than 3 Minutes

Your IBM Model 50, 60, or 75  
Electronic Typewriter  
can be an RS232C PRINTER or TERMINAL



CALIFORNIA MICRO COMPUTER Models 5060 and 5061 can be installed easily and require NO modifications to the typewriter.

For additional information contact:

CALIFORNIA MICRO COMPUTER  
9323 Warbler Ave., Fountain Valley, CA.  
92708 (714) 968-0890

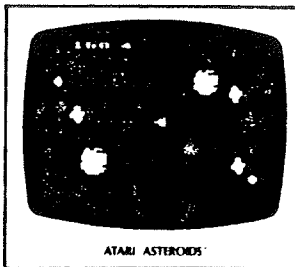


# ATARI

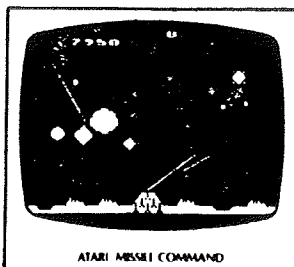
Please see page 229 for further information

Turn to our  
Double Page  
Advertisement  
for SUPER  
CHRISTMAS  
SAVINGS!

## THE PERFECT PRESENT!



ATARI ASTEROIDS



ATARI MISSILE COMMAND

WE CARRY THE COMPLETE LINE OF ATARI SOFTWARE,  
PERIPHERALS AND ACCESSORIES. CALL FOR THE SUPER  
CHRISTMAS SAVINGS!!



**WEST COAST**  
1-800-235-3581

**EAST COAST**  
1-800-556-7586

the robot will leave the rocket and try to enter the building. To do this it needs the key, so it will go to the cave to get it. Once it is in the building, it will drop the key and pick up the fuel. Finally, it will return to the rocket with the fuel. At this point, the PROLOG statements describing the "environment" will be as shown in listing 4. Note that the airlock and the door to the building are left open because the robot did not bother to close them.

This robot is not very bright. If, starting with the initial situation, for instance, you ask the robot to move the gold from the cave to the building (fetch(gold, building)?), it will go to the cave, pick up the gold, and go to the building. At this point the robot realizes it needs the key to open the door, so it returns to the cave to get the key. Since it can carry only one thing at a time, it drops the gold and picks up the key. It then returns to the door, opens it, and enters the building. It now thinks it has succeeded in moving the gold to the building, but the gold is still sitting in the cave where the robot dropped it. This problem is caused by the fact the robot may undo part of the overall goal by backtracking to accomplish a subgoal.

### A Modest Proposal

It would not be too difficult to make the robot intelligent enough to handle the above problem. But instead of making the robot more intelligent, let's give it some "consciousness." Any robot worth its positronic brain must obey the three laws of robotics as postulated by Isaac Asimov (see reference 1). These laws are:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey orders given it by human beings except where such orders would conflict with the first law.
3. A robot must protect its own existence as long as such protection does not conflict with the first or second law.

In order to obey these laws, a robot must not simply obey commands blindly. It must first determine whether it can perform the command without violating the laws.

Listing 4: Status of the hypothetical space expedition's environment after the robot has accomplished its tasks.

```
inside(human, rocket).
inside(robot, rocket).
inside(fuel, rocket).
inside(gold, rocket).
inside(key, building).
entrance(airlock, rocket).
entrance(door, building).
entrance(hole, cave).
liftable(key).
liftable(fuel).
liftable(gold).
```



**Listing 5:** *Asimov's three rules of robotics as implemented in PROLOG. These rules allow the robot to protect humans by shooting aliens, and even by injuring itself, should the situation warrant. The application of the "mini-interpreter" obey makes this a simple proposition for PROLOG.*

```

c  ((%p,%s)) ← !,obey(%p),obey(%s).
obey((%p;%s)) ← !,obey(%p);obey(%s).
obey(%goal) ← clause(%goal,%subgoals),protect(human),protect(robot),
             obey(%subgoals),!.
protect(%x) ← (in_danger(%x,%danger),eliminate(%danger));true.
in_danger(%x,alien) ← not(injured(alien)),inside(alien,%place),
                    inside(%x,%place).
eliminate(%danger) ← shoot(%danger).
shoot(%x) ← %x <> human,inside(%x,%place),moveto(%place),assert(injured(%x)).

```

This is easy to do in PROLOG (statements in listing 5 are explained individually below). Rather than issuing a command such as "fetch(fuel,rocket)?" you must now tell the robot to:

```
obey(fetch(fuel,rocket))?
```

"Obey" is a "mini-interpreter" for PROLOG that checks to see whether the human or the robot needs protecting before executing the subgoals associated with a goal. (Comments in PROLOG are surrounded by /\* and \*/.) For example:

```
/* If a command consists of two subcommands,
   execute them one after the other*/
```

```
obey((%p,%s)) ← !,obey(%p),obey(%s).
```

The exclamation point is a signal to PROLOG that if backtracking causes a return to that point, then the parent goal should be failed immediately, rather than trying to find another solution. This is used here to insure that "obey" does not introduce any extra backtracking.

```
/* If the command consists of a choice between
   two commands, execute one or the other of them */
```

```
obey((%p;%s)) ← !,obey(%p);obey(%s).
```

If there is only one command and also a statement that matches it, protect the human and robot and then execute the subgoals associated with the goal. Note that *clause* is a built-in function. "Clause(%goal,%subgoals)" will return the subgoals associated with a goal):

```
obey(%goal) ← clause(%goal,%subgoals),
             protect(human),protect(robot),obey(%subgoals),!.
```

```
/* If the command is a built-in function, execute it */
obey(%p) ← %p,!.

```

```
protect(%x) ← (in_danger(%x,%danger),
              eliminate(%danger));true.

```

Now let there be an alien in the building who, as long as he is not injured, will attempt to injure anything in the same place as he is:

```
inside(alien,building).
```

```
in_danger(%x,alien) ← not(injured(alien)),
                    inside(alien,%place),inside(%x,%place).
```

Assume also that the robot has a phasor and will use it to eliminate danger:

```
eliminate(%danger) ← shoot(%danger).
```

Anything that is shot is injured. However, under no circumstances will the robot shoot a human:

```
shoot(%x) ← %x <> human,inside(%x,%place),
            moveto(%place),assert(injured(%x)).

```

Now if you tell the robot the fetch the fuel to the rocket:

```
obey(fetch(fuel,rocket))?
```

the robot enters the building and shoots the alien in order to protect itself. The robot then carries the fuel to the rocket. If you ask the robot to shoot the human:

```
obey(shoot)human)?
```

the robot will not obey because that would violate the first law. However, if you ask the robot to shoot itself:

```
obey(shoot)robot)?
```

it will do so because the second law of robotics takes precedence over the third.

I hope that this brief introduction has given you an indication of the simplicity and power of nonprocedural languages such as PROLOG. Such languages may represent the next step in the evolution of programming languages. ■

#### References/Suggested Reading

1. Asimov, Isaac. *I, Robot*. New York: Doubleday, 1957.
2. Kowalski, R. *Logic for Problem Solving*. New York: Elsevier-North Holland Publishing Co, 1979.
3. Pereira, L. F. Pereira, and D. Warren. "User's Guide to DECsystem-10 PROLOG," 1978.