Solving Two-State Logic Problems with Boolean Arrays: An Approach Unique to APL

Kenneth Fordyce
Software Quality and Customer Satisfaction
Department D14B, Mail Station 922
International Business Machines Corp.
Kingston, New York 12401 USA

Manuel Alfonseca
Senior Technical Staff
International Business Machines Corp.
Madrid Scientific Center
Paseo de la Castellana, 4
28043 Madrid, Spain

James Brown
APL Chief Architect
International Business Machines Corp.
Santa Teresa Development Lab
J88 / E42, POB 50020
San Jose, California 95150 USA

Gerald Sullivan
Senior Engineer
Manager, Advanced Industrial Engineering
and LMS Development
International Business Machines Corp.
Department 746, Building 965-3
Essex Junction, Vermont 05452 USA

Abstract

Many great advances in science and mathematics were preceded by notational improvements. While a given algorithm can be implemented in any general purpose programming language, discovery of algorithms is heavily influenced by the notation used to investigate them. It is often conjectured that APL notation leads to unique solutions. In logic problems the values for variables are often limited to two states or options: yes/no, true/false, on/off, guilt/innocence, etc. This paper provides one example of how APL provides a unique solution to such problems.

Example: *Who is Guilty?*

The following logic problem is adapted from Raymond Smullyan's book *"Alice in Puzzle-Land."*

The jam had been stolen by either the March Hare, the Mad Hatter, or the Dormouse. They were arrested and each made one statement. They were:

- The March Hare: *I am not guilty.*
- The Mad Hatter: *I am not guilty.*
- The Dormouse: *At least one of the others speaks the truth.*

Further investigation produced the following conclusions (facts known to be true).

1. The March Hare and the Dormouse didn't both say the truth.
2. Only one of them is guilty.

Who is guilty?

To solve this problem we establish three logical variables.

- The Dormouse is guilty (a 1) or not guilty (a 0) (*DG*).
- The Mad Hatter is guilty (a 1) or not guilty (a 0) (*HG*).
- The March Hare is guilty (a 1) or not guilty (a 0) (*MG*).

These can be represented by the following three vectors:

```
    DG←0 1 0 1 0 1 0 1
    HG←0 0 1 1 0 0 1 1
    MG←0 0 0 0 1 1 1 1
```

These three variables represent all possible combinations of guilt and innocence. For example, Column 1 (*DG* = 0, *HG* = 0, *MG* = 0) represents the option all 3 are not guilty. Column 3 (*DG* = 0, *HG* = 1, *MG* = 0) represents the option the Dormouse and the March Hare are not guilty, but the Mad Hatter is guilty.

Fact 1 tells us that the March Hare and the Dormouse didn't both tell the truth—at least one is lying.

The March Hare stated he was not guilty. If this is true then *MG* has the value 0. This statement can be expressed as:

```
    0 = MG
```

The Dormouse stated at least one of the others is not guilty. If this is true, then at least one of the others is not guilty and has the value 0. Therefore *and*ing (∧) *HG* with *MG* will yield a zero. This statement can be expressed as:

```
    0 = (HG ∧ MG)
```

Since we know at least one of these statements is false, then *and*ing (∧) the two prior statements together must yield a 0. Therefore, only combinations of guilty (1) and not guilty (0) that yield a zero when these two expressions are "and"-ed are still possible solutions to the crime.

```
    0 = ((0=MG) ∧ (0=(HG ∧ MG)))
0 0 0 0 1 1 1 1
```

Where there is a 1, those combinations of guilty and not guilty are still possible solutions.

Fact 2 states that only one and only one of the three is guilty. This statement can be expressed as:

```
    1 = (HG + MG + DG)
0 1 1 0 1 0 0 0
```

Possible solutions to the crime are limited to combinations of guilty (1) and not guilty (0) making the above statement true (yield a 1).

Since we know Fact 1 and Fact 2 are both true, then *and*ing them together must yield a 1. This statement can be expressed as:

```
    1=(0=(0=MG)∧(0=HG∧MG))∧(1=HG+MG+DG)
0 0 0 0 1 0 0 0
```

Again, only combinations of guilty (1) and not guilty (0) that make the above statement true (yield a 1) are possible solutions to the crime.

In this case only combination 5 makes this statement true. Therefore we conclude that *the March Hare is guilty.* ■

A Kaizen Strategy for APL Education

Dick Holt
HRH Systems, POBox 4496
Silver Spring, MD 20914 USA

If APL is to survive, more people must be able to quickly learn how to use it in their work or business. In the past year, powerful new APL learning material has become available that's cheap or free, fast, and with big workspaces. This new material also offers quick and easy upgrade paths to commercial APLs.

A common theme of this fresh material is that it's continuously and incrementally improved. The Japanese term for this process is *"kaizen."* It is *kaizen*, in part, that explains the hammering that Japan is delivering to the auto industry world-wide. *Kaizen* develops product innovation at the same time that it generates cash flow to pay for further innovation. *Kaizen* is based on a quick feedback loop from innovation, to market, and back to innovation again (have you bought a software upgrade recently?)

The *kaizen* loop is apparent in IBM's steady and cumulative improvement of TryAPL2—six enhancements in two years. Even the once-moribund Sharp APL/PC has been vividly rejuvenated by Iverson's 386 enhancements. Manugistics (formerly STSC) has a habit, almost to a fault, of frequent incremental innovation. And, the APL lessons from the Capital PC User Group (CPCUG) have been incrementally improved in multiple ways, in two generations beyond Z.V. Jizba's pioneering effort.

Kaizen strategy is a challenge to APL educators: people who are serious about the future of APL must move APL out of the schools and into the market place. Today, most people learn more on the job than they do in schools. They have no choice—technology changes too fast. Unless the time between learning APL and applying it at work is short, technology moves ahead rapidly, and the learner misses the boat.

George Bernard Shaw once compared photographers to spawning salmon: they take a million pictures in the hopes that one will turn out well. Teaching APL in schools is no different—like spawning salmon, it's tantamount to playing the lottery.

In its APL classes last summer, the CPCUG found that only one of its 14 learners was a student. All others were professionals seeking to improve their productivity at work. Six of the 14 did not have English as their native tongue. This typifies today's dynamic global market environment within which APL must compete.

Here's a source list of APL learning material, all new or improved within the past year:

- IBM's TryAPL2: Free demo disk of Version 2.0 available from:

  IBM APL Development
  M46/D12-278B, Santa Teresa Lab
  Box 49023, San Jose CA 95161-9023
  On-screen User Manual and Lessons, LJ printable.

- Sharp APL/PC: Sharp APL/PC is available from:

  Iverson Software Inc.
  33 Major Street, Suite 466
  Toronto, Ontario
  Canada M5S 2K9
  Tel: 416-925-6096
  Fax: 416-488 7559

- Documentation and enhancements to Sharp (color, faster 386 version, editor, windows) are available from Iverson. This material isn't free, but it's not expensive either.

- Manugistics: Free run-time demo of APL★PLUS/PC, with a well-designed interactive keyboard diagram that explains APL symbols and gives examples of their use.

- Call Manugistics at 800-592-0050, or 301-984-5123 in MD, or 301-984-5412 in the Washington, D.C. area. Outside the US, contact your local dealer, or call 301-984-5412. Or write to:

  Manugistics, Suite 1729
  2115 East Jefferson Street
  Rockville MD 20852