is the only science which has made no progress since the revival of letters; logic is the only science which has produced no growth of symbols." In my view APL is in the best tradition of Boole, De Morgan, Jevons, and Venn.[73]

One of the most striking features in Iverson's *A Programming Language* is his demonstration that "the generalized matrix product and the selection operations together provide an elegant formulation in several established areas of mathematics. A few examples will be chosen from two such areas, symbolic logic and matrix algebra."[74] Iverson proceeded to show how his notation leads to a natural extension of De Morgan's laws.[75]

De Morgan's law:

```
A ∧ B ↔ ~(~A) ∨ ~B
```

Iverson's extensions:

```
∧/U ↔ ~∨/~U
≠/U ↔ ~=/~U
```

In ordinary APL:

```
      U←? 5 4 3 ⍴ 2
      V←? 3 6 7 ⍴ 2

      ∧/, (U ≠.∧ V)=~ (~U) =.∨ (~V)
1
```

In J, the latest form of Iverson's notation, his 1962 example is executed as follows:

```
      u=.?5 4 3 $2
      v=.?3 6 7 $2

      (u~:/ .*.v)-:-.(-.u) =/ .+.(-.v)
1
```

where:

~: is NOT EQUAL; *. is AND; -: is MATCH; -. is NOT; and +. is OR.

In algebra a leading negative can be removed by changing the signs of all quantities in the expression that follows; in APL a leading NOT (~) can be removed by interchanging the pairs AND and OR, EQUALS and NOT-EQUALS, etc. In the following example both functions *F* and *G* remove redundant blanks from a string.

Ordinary APL:

```
      ∇Z←F S;U
[1]     Z←(~U∧1⌽U←S=' ')/S
[2]   ∇

      ∇Z←G S;U
[1]     Z←(U∨1⌽U←S≠' ')/S
[2]   ∇
```

Direct definition:

```
F:(~ U∧1⌽U←⍵=' ')/⍵
G:(U∨1⌽U←⍵≠' ')/⍵
```

APL continues to grow in power, and Iverson's final example,[76] written but not executable as +./ in APL, can be executed in J as follows.

Given:

```
A=. 1 3 2 0, 2 1 0 1,: 4 0 0 2
B=. 4 1, 0 3, 0 2,: 2 0
f=. ~:&0
h=. +/ @ #"0
```

Then:

```
(f A) +/ .h B
4 6
6 4
6 1
```

Iverson's generalized matrix product found immediate application in his formal description of indexed addressing on the IBM 7090 computer,[77] which in one line made clear what takes half a page of text in the *Principles of Operation* manual for that machine. There are, of course, many similar examples in Reference 78.

Arrays and locative symbols

APL is often referred to as the *array processing language*, and its power does to a great extent come from its ability to work with arrays directly, a feature of increasing importance as vector processors and parallel computing become available. When we specify a place by giving its latitude and longitude, or define a point on a scatter diagram by giving its X and Y coordinates, we intend that two numbers should be taken together to identify one object. This is the first step in thinking in terms of what Sylvester called *multiple quantity*.

Stevinus was the first to show how forces combine in the manner we know as the parallelogram of