LEWIS CARROLL'S FIVE LIARS PUZZLE

William Bricken

June 2009

This puzzle appears as the last problem in Lewis Carroll's book, Symbolic Logic.

```
=========================================================================
Five people {A,B,C,D,E} each make two statements.  Determine who tells two
Truths, who tells two Lies, and who tells one Truth and one Lie.

      A    a1:    "Either B or D tells a Truth and a Lie.
           a2:     Either C or E tells two Lies."

      B    b1:    "Either A or C tells a Truth and a Lie.
           b2:     Either D tells two Lies or E tells two Truths."

      C    c1:    "Either A or D tells two Truths.
           c2:     Either B tells a Truth and a Lie or E tells two Lies."

      D    d1:    "Either A or E tells two Lies.
           d2:     Either B tells two Lies or C tells two Truths."

      E    e1:    "Either A or B tells two Truths.
           e2:     Either C or D tells a Truth and a Lie."
=========================================================================
```

The puzzle cleverly hides reference within apparently indirect self-reference, and does not actually incorporate "vicious" self-reference, i.e. X=(X), so imaginary solutions are not necessary.

With only ten variables, exhaustive search for the truth value of each variable is both efficient and available. There are 2^10 (i.e. 1024) possible sets of values, and a truth table with 1024 rows and about 60 columns would specify all possible cases. In a solution, the truth table assignments for each of the ten variables matches the value of the expressions associated with each variable.

The Boolean minimization problem is to solve the puzzle without case analysis, without guessing. The sequel describes an algebraic solution that does not call upon case analysis.

We begin by encoding the ten assertions {a1,a2,b1,b2,c1,c2,d1,d2,e1,e2} using the following map to boundary logic:

```
      one Truth and one Lie         X≠Y         ==>  ((X Y)((X)(Y)))
      two Lies                     (X Y)
      two Truths                  ((X)(Y))
      Either...or                   X≠Y
```

Note that inequality is a composite of the forms for two Truths and two Lies, essentially reading "not two Lies or two Truths". Equality statements can be encoded in shorthand by X=Y, or in parens form as (X Y) ((X)(Y)).

SYMBOLIC ENCODING

```
      a1:       b1≠b2 .≠. d1≠d2
      a2:      (c1 c2) ≠ (e1 e2)
      b1:       a1≠a2 .≠. c1≠c2
      b2:      (d1 d2) ≠ ((e1)(e2))
      c1:   ((a1)(a2)) ≠ ((d1)(d2))
      c2:       b1≠b2 .≠. (e1 e2)
      d1:      (a1 a2) ≠ (e1 e2)
      d2:      (b1 b2) ≠ ((c1)(c2))
      e1:   ((a1)(a2)) ≠ ((b1)(b2))
      e2:       c1≠c2 .≠. d1≠d2
```

For convenience, representation of the ten statements is next standardized using equality (rather than inequality), with negation pushed outside of the equality relation, i.e. X≠Y is written as (X=Y).

STANDARDIZATION USING EQUALITY RELATIONS

```
      a1:     (b1=b2 .=. d1=d2)
      a2:      (c1 c2 = e1 e2)
      b1:     (a1=a2 .=. c1=c2)
      b2:      (d1 d2 = (e1)(e2))
      c1:   ((a1)(a2) = (d1)(d2))
      c2:     (b1=b2 .=. e1 e2)
      d1:      (a1 a2 = e1 e2)
      d2:      (b1 b2 = (c1)(c2))
      e1:   ((a1)(a2) = (b1)(b2))
      e2:     (c1=c2 .=. d1=d2)
```

We are heading for a single parens expression of the puzzle. The outermost equality relation for each statement is next expanded to parens form. We temporarily keep the notation for equality between variables (such as b1=b2), for ease of reading.

THE OUTER EQUALITIES EXPANDED

```
      a1:   ( (b1=b2 d1=d2) ((b1=b2)(d1=d2)) )
      a2:   ( (c1 c2 e1 e2) ((c1 c2)(e1 e2)) )
      b1:   ( (a1=a2 c1=c2) ((a1=a2)(c1=c2)) )
      b2:   ( (d1 d2 (e1)(e2)) ((d1 d2)((e1)(e2))) )
      c1:   ( ((a1)(a2)(d1)(d2)) (((d1)(d2)) ((a1)(a2))) )
      c2:   ( (b1=b2 e1 e2) ((b1=b2)(e1 e2)) )
      d1:   ( (a1 a2 e1 e2) ((a1 a2)(e1 e2)) )
      d2:   ( (b1 b2 (c1)(c2)) ((b1 b2)((c1)(c2))) )
      e1:   ( ((a1)(a2)(b1)(b2)) (((a1)(a2))((b1)(b2))) )
      e2:   ( (c1=c2 d1=d2) ((c1=c2)(d1=d2)) )
```

HOW TO DEAL WITH LIARS

The difficulty posed by this type of puzzle is that each statement may be either True (spoken truthfully) or False (a lie). So the conventional technique of using conjunction to combine (True) statements-as-assertions is not available.

The key to combining possibly True or False premises is that each statement is indeed equal to the variable that names it. The above notation for naming the statements can be expressed in the form of True equations:

```
            "X:  Y"    means     X=Y  is True.
```

We can thus convert the ten indeterminate statements into ten True statements (assertions) about the labels of the statements.

INDETERMINATE STATEMENTS MADE INTO ASSERTIONS

```
      a1 = ( (b1=b2 d1=d2) ((b1=b2)(d1=d2)) )
      a2 = ( (c1 c2 e1 e2) ((c1 c2)(e1 e2)) )
      b1 = ( (a1=a2 c1=c2) ((a1=a2)(c1=c2)) )
      b2 = ( (d1 d2 (e1)(e2)) ((d1 d2)((e1)(e2))) )
      c1 = ( ((a1)(a2)(d1)(d2)) (((d1)(d2)) ((a1)(a2))) )
      c2 = ( (b1=b2 e1 e2) ((b1=b2)(e1 e2)) )
      d1 = ( (a1 a2 e1 e2) ((a1 a2)(e1 e2)) )
      d2 = ( (b1 b2 (c1)(c2)) ((b1 b2)((c1)(c2))) )
      e1 = ( ((a1)(a2)(b1)(b2)) (((a1)(a2))((b1)(b2))) )
      e2 = ( (c1=c2 d1=d2) ((c1=c2)(d1=d2)) )
```

Now, these ten assertions can be combined via conjunction into a single expression that is also True.

A SINGLE ASSERTION

Next, the outer "label" equalities are converted into parens structure, as are the inner equalities between variables. The resulting form (on the next page) is a single parens expression, within which the ten assertions are structurally visible.

In this form, there are no concepts of indirect reference or of self-reference. Instead, the nesting of parens and the relative location of the ten variables incorporates all the information within the original puzzle.

The resulting expression is not human friendly, but it can be processed by the Losp Boolean minimization engine to yield the solution to the puzzle. The parens expression that represents the puzzle is relatively small and is reduced within a few seconds. Incidentally, I could not solve this problem by hand.

THE PARENS FORM OF THE PUZZLE

```
      ( ((a1 (((b1 b2)((b1)(b2)) (d1 d2)((d1)(d2)))
                  (((b1 b2)((b1)(b2)))((d1 d2)((d1)(d2))))))
            ((a1)((((b1 b2)((b1)(b2)) (d1 d2)((d1)(d2)))
                  (((b1 b2)((b1)(b2)))((d1 d2)((d1)(d2)))))))

        ((a2 ((c1 c2 e1 e2) ((c1 c2)(e1 e2))))
            ((a2)(((c1 c2 e1 e2) ((c1 c2)(e1 e2))))))

        ((b1 (((a1 a2)((a1)(a2)) (c1 c2)((c1)(c2)))
                  (((a1 a2)((a1)(a2)))((c1 c2)((c1)(c2))))))
            ((b1)((((a1 a2)((a1)(a2)) (c1 c2)((c1)(c2)))
                  (((a1 a2)((a1)(a2)))((c1 c2)((c1)(c2)))))))

        ((b2 ((d1 d2 (e1)(e2)) ((d1 d2)((e1)(e2)))))
            ((b2)(((d1 d2 (e1)(e2)) ((d1 d2)((e1)(e2)))))))

        ((c1 (((a1)(a2)(d1)(d2)) (((d1)(d2)) ((a1)(a2)))))
            ((c1)((((a1)(a2)(d1)(d2)) (((d1)(d2)) ((a1)(a2)))))))

        ((c2 (((b1 b2)((b1)(b2)) e1 e2) (((b1 b2)((b1)(b2)))(e1 e2))))
            ((c2)((((b1 b2)((b1)(b2)) e1 e2) (((b1 b2)((b1)(b2)))(e1 e2))))))

        ((d1 ((a1 a2 e1 e2) ((a1 a2)(e1 e2))))
            ((d1)(((a1 a2 e1 e2) ((a1 a2)(e1 e2))))))

        ((d2 ((b1 b2 (c1)(c2)) ((b1 b2)((c1)(c2)))))
            ((d2)(((b1 b2 (c1)(c2)) ((b1 b2)((c1)(c2)))))))

        ((e1 (((a1)(a2)(b1)(b2)) (((a1)(a2))((b1)(b2)))))
            ((e1)((((a1)(a2)(b1)(b2)) (((a1)(a2))((b1)(b2)))))))

        ((e2 (((c1 c2)((c1)(c2)) (d1 d2)((d1)(d2)))
                  (((c1 c2)((c1)(c2)))((d1 d2)((d1)(d2))))))
            ((e2)((((c1 c2)((c1)(c2)) (d1 d2)((d1)(d2)))
                  (((c1 c2)((c1)(c2)))((d1 d2)((d1)(d2))))))) )
```

THE LOSP ENGINE

The Losp engine recursively applies the three boundary logic reduction rules to parens expressions:

```
                (A ( )) =                 Occlusion
                  ((A)) =  A              Involution
                  A {A} = A { }           Pervasion
```

Curly braces stand in place of any intervening structure. No other transformations are used to minimize the parens representation of the problem, although the application of Pervasion is at times subtle.

THE RESULT

The engine returns the following minimization, with variable equalities written in shorthand:

```
  (C1 D1 D2 (B2) (A2 C2) (E1 E2) ((A1)(A2)) ((B1)(C2)) b1=e1  e1=e2 (A1 ((A2)(B1)))))
```

Since this form is True by construction, all top-level subforms must be False, otherwise the expression would collapse to nothing via Occlusion:

```
      (X Y Z) =T     means     X=F, Y=F, and Z=F      since     (X=T Y Z) =F
```

Technically, the contents of the outermost parens must be void-equivalent.

Variable bindings are available by setting the value of each top level sub-expression to False. For convenience, we standardize compound statements to assertions of truth:

c1	=F			
d1	=F			
d2	=F			
(b2)	=F	==>	b2	=T
(a2 c2)	=F	==>	a2 c2	=T
e1=e2	=F	==>	e1≠e2	=T
((a1)(a2))	=F	==>	(a1)(a2)	=T
((b1)(c2))	=F	==>	(b1)(c2)	=T
(a1 ((a2)(b1)))	=F	==>	a1 ((a2)(b1))	=T
b1=e1	=F	==>	b1≠e1	=T

The first four variables have forced values. The remaining variables are mutually contingent, which implies that there is more than one solution. We can arbitrarily assign one variable the two possible truth values (T and F), and follow the algebraic consequences. (This is not case analysis since both choices are solutions.)

Let e1=T:

e1≠e2	=T	==> e2=F
b1=e1	=T	==> b1=T
(b1)(c2)	=T	==> c2=F
a2 c2	=T	==> a2=T
(a1)(a2)	=T	==> a1=F
a1 ((a2)(b1))	=T	==> confirmed

Let e1=F:

```
e1≠e2          =T         ==> e2=T
b1=e1          =T         ==> b1=F
a1 ((a2)(b1))=T           ==> a1=T
(a1)(a2)       =T         ==> a2=F
a2 c2          =T         ==> c2=T
(b1)(c2)       =T         ==> confirmed
```

Exploring the possible values for one variable (we arbitrarily chose e1 above) determines all other values. That is, there are exactly two solutions. Note that in the two solutions, every remaining variable must take a different value for each of the solutions; if any remaining variable had only one possible value, it would have been isolated (like c1) during the reduction. The assignment of value to e1 above leaves five unknowns expressed within six assertions, so one assertion will be redundant. Interestingly, unlike algebraic equations, the six equations with six unknowns do not determine the values of the variables.

Summarizing the determined and contingent values in both solutions:

Solution I: {c1,d1,d2} = F, {b2} = T, {a1,c2,e2} = F, {a2,b1,e1} = T

Solution II: {c1,d1,d2} = F, {b2} = T, {a1,c2,e2} = T, {a2,b1,e1} = F

Converting these results into statements about each of the five people:

	Solution I	Solution II
Two Truths:	B	
A Truth and a Lie:	A,E	A,B,C,E
Two Lies:	C,D	D

CONCLUSION

Algebraic treatment of this type of puzzle shows that networks of reference are safe in logic, so long as one type of equality assertion (and its transitive consequences) is flagged, X=(X). Computation in boundary logic flags imaginary self-equality by making it void-equivalent: when a variable disappears from a solution, it is either false or imaginary. We might say that imaginary solutions are resurrected from the void. One available global assignment of values is to make all variables imaginary. It that case, the problem itself disappears.