A Program for Multi-Valued Logic

Ulisses Ferreira

***Abstract*—**This paper introduces a program for helping a researcher find a several-valued logic given the number of values, tables, optional rules and desired properties. It is based on constraint programming and can be easily adapted. As an example, the program works on a six-valued logic.

***Keywords*—**Multiple-valued logics, 3-valued logic, 6-valued logic, six-valued logic, knowledge based systems.

I. Introduction

In 2006, the present author introduced a 7-valued logic[1] given a model for a logical type for Plain programming language design. In that logic, the *kk* value can refer to contingency in a consistent setting, i.e. something might happen or might not happen. That means that a variable can be either false or true but not both, and that can remain *kk* for long, or until the end of the run or persistently forever. On the other hand, the *uu* value can also apply to contingency but in a setting that permits inconsistent condition, i.e. something might happen or might not happen, and it is also possible that *uu* becomes both false and true, that is, inconsistent (the inconsistent value is denoted as *ii*). For instance, a person X that decides to charge a person Y at the Court: X shows his or her proofs against Y, whereas Y shows his or her proofs in favour of himself or herself, or even against X. Both X and Y try to convince the Judge but in opposite directions. In the end, the Judge may feel unable to decide because he or she has received too much information in both directions. Thus, there are two settings: consistent and inconsistent. Furthermore, it is possible that one setting can be changed to become the other setting in the same run, in particular, when science discovers or invent something, and then the setting changes from subjective to objective, from personal opinions to an absolute truth. The laws also change during run time although it is rare to find an example of application. Nonetheless, at the logical and programming language design levels, that should not be a concern, but to aim at being a more general as possible. Perhaps an example was the discovery of the DNA test, which started deciding many cases at Court when the father did not pay income for a baby if he stated that it was not his son or daughter. In that case, the setting changed from the possibly inconsistent to a certainly consistent. The referred to 7-valued logic proposes both settings, for the work on programming language design needs to offer a tool for general purpose. In this sense, while every application or function or variable can be either monotonic or non-monotonic, the programming language that support them should offer both. In the logic, at the basic level, the consistent setting has the two Boolean values whereas the inconsistent setting has three values: *ff*, *ii*, *tt*, for representing false, inconsistent and true, respectively. At one level above, there might be *kk* as stated, *fi* (either false or inconsistent) and also *it* (either inconsistent or true). So the latter three values represent possible ranges of the basic level. Finally, at the third and top level, there is *uu* (unknown), which is the value that represents the broadest range of possibilities and *uu* is normally the initial value for a variable, although there are exceptions. From unknown, the range might shorten to either *fi* (either false or inconsistent) or to *it* (either inconsistent or true), whereas, from *fi*, the range might shorten either to false or to inconsistent, and whereas, from the *it*, the range might shorten either to inconsistent or to true. Thus, there are two alternative paths from *uu* to *ii*. Thus, *kk* is the 7th value. To avoid mistakes, the symbol of two letters together 'ft' is not used for standing for 'either false or true', because that name could have been interpreted not only as *kk*, but also as *uu*, i.e. either as 'either false or true', hence a consistent value (*kk*), or as a range 'from false until true' (*uu*). The present paper makes use of 6-valued logics only, for simplifying the work and make the runs much faster. That is, *kk* is outside the present work towards a 6-valued logic.

The negation of *ii* is *ii*, the negation of *uu* is *uu*. That is, $\neg ii = ii, \neg uu = uu$. Given these negations, $(a \wedge a) \leftrightarrow a$ in this paper also for ii and uu. So, the law of the excluded middle, i.e. $\neg(a \vee \neg a)$ or in Peirce's form, $\big((\mathrm{a} \rightarrow \mathrm{b}) \rightarrow \mathrm{a}\big) \rightarrow \mathrm{a}$), and the law of non-contradiction, i.e. $\neg(a \wedge \neg a)$, are not tautologies. The law of excluded middle of the classical logic has been rejected in bivalent logics including constructive one. That property sometimes is not checked here. Properties such as contrapositive, i.e. $(a \rightarrow b) \leftrightarrow (\neg b \rightarrow \neg a)$, became harder because of this apparently somewhat odd negation, but the same negation has been present since both Lukasiewicz[2] and Kleene[3] 3-valued logics. Both negations can be seen as two different signals of exception. If a predicate in a knowledge base contains inconsistency, it is good for its deductive system to be able to inform that condition at the top level if necessary, and even its user in any query. Thus, its negation can be seen as propagation of the same condition. However, expressions such as $a \wedge ff$ must result in ff regardless the a value. Yet, values mean ranges and the negation of *fi* is *it* whereas the negation of *it* is *fi* hence the double negation property holds. Certainly, all Boolean expressions provide Boolean results wherever all operands are Boolean. Another approach for the negation is in Belnap 4-valued logic[4] and recently in [5], where the negation of *ii* is *uu* whereas the negation of *uu* is *ii*. The present work also considers that $a \rightarrow a$ is a tautology. Otherwise, the identity property would not hold. $ff \rightarrow a$, $a \rightarrow tt$ are also tautologies here.

The program run reads the user's tables: negation,

The author's affiliation is the Universidade Federal da Bahia (e-mail: jose@ulissesferreira.uk).

conjunction, partial implication and equivalence. The disjunction table is built by calculating its values from the negation and the conjunction tables by using both De Morgan's Laws. In a table entry in the input file, instead of a two-letter value, a sequence e of dots together, i.e '..', tells the program that the corresponding entry of the table is left for the program to work. So, the execution of the program completes the conjunction table in such a manner to make the commutative and associative properties and both De Morgan Laws hold. In a kind of a short dialog, it also works on the user's implication table where the user leaves open. After the declaration of the tables, it is also possible for the user to include rules in the input file after a line containing the word `rules` and before a line containing the word `end`. Each rule is in the form `v1 -> v2` in the same line, where `v1` and `v2` are in $\{ff, fi, ii, uu, it, ff\}$. Each rule adds a possibility for an entry of the implication table. The double exclamation '!!' followed by a space before a rule tells the program not to consider that specific case of implication in its searching. In the present paper, the examples do not contain such additional rules, neither positive nor negative. The program completes rules and builds a list of constraints before starting searching for logics. Thus, depending on the number of values, the list of constraints makes the run much faster or even tries to make the time feasible, in comparison to the brute force method. Since this simple program does not include neither lexical analyzer nor a parser, the input file has to end with a line containing the word `end` even if there is no rule. Otherwise, the results are not correct.

From a number of values on, there may happen that the program outputs a large list of solutions. For each presented solution and below the table, the program also shows its score, which aims at indicating "how good" that solution is. Currently, it has been regarded that, for every open entry of the implication table, the closer to false the value is, the better the solution is. Thus, the order of the declaration of the values in the program is relevant, from false until true. Between *uu* and *ii*, it is a matter of choice.

Regarding the choice of a programming language, languages such as Java and Plain have their own Virtual Machine, hence their computations are not as fast as languages such as C or C++ whose compilers can generate code for specific hardware. Those object-oriented languages that support late binding are not suitable for any application where the run time is critical. Because of that reason, the program was written in C. Nonetheless, Java programmers can easily understand the introduced program and then can make changes for their purpose.

II. TESTING AND RESULTS

By constraint programming, while testing, all runs have taken less than half an hour. Most of them have resumed within a few minutes. For a 7-valued logic, the time is exponentially longer.

A. With the strongest equivalence, i.e. $(a, a) = tt; (a, b) = ff \; if \; a \neq b$*:*

Checking the properties, the run states that $(uu \rightarrow ff) = \{\}$ and $(ii \rightarrow ff) = \{\}$. No solution and no need for searching;

Without ((a→b)→a)→a), no solution after searching;

Without checking the distributive property among implications, i.e. (a→(b→c))→((a→b)→(a→c)), the run gives no solutions either, and there is no need for searching. The print of the run is the following as an example:

```
oem@oem-linux:~/amvlog$ gcc a6vlog.c -o u
oem@oem-linux:~/amvlog$ ./u
Welcome to a run searching for a 6-valued logic.
Warning: Sorry, the current version does not permit search over conjunctions. In the present case, (ff and fi) has no value.
Would you like my suggestions for such open values (y/n)? y
Suggested value: ff and fi = ff
Suggested value: ff and uu = ff
Suggested value: ff and ii = ff
Suggested value: ff and it = ff
Suggested value: ff and tt = ff
Suggested value: fi and uu = fi
Suggested value: fi and ii = fi
Suggested value: fi and it = fi
Suggested value: fi and tt = fi
Suggested value: uu and ii = fi
Suggested value: uu and it = uu
Suggested value: uu and tt = uu
Suggested value: ii and it = ii
Suggested value: ii and tt = ii
Suggested value: it and tt = it

Negation:
   | ff fi uu ii it tt
---+------------------
   | tt it uu ii fi ff

Conjunction:
   | ff fi uu ii it tt
---+------------------
ff | ff ff ff ff ff ff
fi | ff fi fi fi fi fi
uu | ff fi uu fi uu uu
ii | ff fi fi ii ii ii
it | ff fi uu ii it it
tt | ff fi uu ii it tt
```

```
  Disjunction:
     | ff fi uu ii it tt
  ---+------------------
  ff | ff fi uu ii it tt
  fi | fi fi uu ii it tt
  uu | uu uu uu it it tt
  ii | ii ii it ii it tt
  it | it it it it it tt
  tt | tt tt tt tt tt tt

  Equivalence:
     | ff fi uu ii it tt
  ---+------------------
  ff | tt ff ff ff ff ff
  fi | ff tt ff ff ff ff
  uu | ff ff tt ff ff ff
  ii | ff ff ff tt ff ff
  it | ff ff ff ff tt ff
  tt | ff ff ff ff ff tt

  Implication:
     | ff fi uu ii it tt
  ---+------------------
  ff | tt tt tt tt tt tt
  fi | .. tt .. .. .. tt
  uu | .. .. tt .. .. tt
  ii | .. .. .. tt .. tt
  it | .. .. .. .. tt tt
  tt | ff fi uu ii it tt

  Any rules?
  That is ok, I have found no rule in the input data.
  Error: I am afraid, there are one or more impossibilities. Check the following:
  (uu -> ff) = { }
  (ii -> ff) = { }
  The whole set of constraints:
  (ff -> ff) = { tt }
  (ff -> fi) = { tt }
  (ff -> uu) = { tt }
  (ff -> ii) = { tt }
  (ff -> it) = { tt }
  (ff -> tt) = { tt }
  (fi -> ff) = { it }
  (fi -> fi) = { tt }
  (fi -> uu) = { tt }
  (fi -> ii) = { tt }
  (fi -> it) = { tt }
  (fi -> tt) = { tt }
  (uu -> ff) = { }
  (uu -> fi) = { fi ii it tt }
  (uu -> uu) = { tt }
  (uu -> ii) = { fi ii it tt }
  (uu -> it) = { tt }
  (uu -> tt) = { tt }
  (ii -> ff) = { }
  (ii -> fi) = { fi uu it tt }
  (ii -> uu) = { fi uu it tt }
  (ii -> ii) = { tt }
  (ii -> it) = { tt }
  (ii -> tt) = { tt }
  (it -> ff) = { fi }
  (it -> fi) = { fi uu ii tt }
  (it -> uu) = { fi uu ii tt }
  (it -> ii) = { fi uu ii tt }
  (it -> it) = { tt }
  (it -> tt) = { tt }
  (tt -> ff) = { ff }
  (tt -> fi) = { fi }
  (tt -> uu) = { uu }
  (tt -> ii) = { ii }
  (tt -> it) = { it }
  (tt -> tt) = { tt }
  Thus, there is no need for searching.
oem@oem-linux:~/amvlog$
```

Without checking the contrapositive property, there is a single solution:

```
  oem@oem-linux:~/amvlog$ gcc amvlog.c -o a6L
  oem@oem-linux:~/amvlog$ ./a6L
  Welcome to a run searching for a 6-valued logic.
  Warning: Sorry, the current version does not permit search over conjunctions. In the present case, (ff and fi) has no value.
  Would you like my suggestions for such open values (y/n)? y
  Suggested value: ff and fi = ff
  Suggested value: ff and uu = ff
  Suggested value: ff and ii = ff
  Suggested value: ff and it = ff
  Suggested value: ff and tt = ff
  Suggested value: fi and uu = fi
  Suggested value: fi and ii = fi
  Suggested value: fi and it = fi
  Suggested value: fi and tt = fi
  Suggested value: uu and ii = fi
  Suggested value: uu and it = uu
  Suggested value: uu and tt = uu
  Suggested value: ii and it = ii
  Suggested value: ii and tt = ii
  Suggested value: it and tt = it

  Negation:
     | ff fi uu ii it tt
  ---+------------------
     | tt it uu ii fi ff
```

```
  Conjunction:
     | ff fi uu ii it tt
  ---+------------------
  ff | ff ff ff ff ff ff
  fi | ff fi fi fi fi fi
  uu | ff fi uu fi uu uu
  ii | ff fi fi ii ii ii
  it | ff fi uu ii it it
  tt | ff fi uu ii it tt

  Disjunction:
     | ff fi uu ii it tt
  ---+------------------
  ff | ff fi uu ii it tt
  fi | fi fi uu ii it tt
  uu | uu uu uu it it tt
  ii | ii ii it ii it tt
  it | it it it it it tt
  tt | tt tt tt tt tt tt

  Equivalence:
     | ff fi uu ii it tt
  ---+------------------
  ff | tt ff ff ff ff ff
  fi | ff tt ff ff ff ff
  uu | ff ff tt ff ff ff
  ii | ff ff ff tt ff ff
  it | ff ff ff ff tt ff
  tt | ff ff ff ff ff tt

  Implication:
     | ff fi uu ii it tt
  ---+------------------
  ff | tt tt tt tt tt tt
  fi | .. tt .. .. .. tt
  uu | .. .. tt .. .. tt
  ii | .. .. .. tt .. tt
  it | .. .. .. .. tt tt
  tt | ff fi uu ii it tt

  Any rules?
  That is ok, I have found no rule in the input data.
  Please press [Enter] to see the set of constraints:
  (ff -> ff) = { tt }
  (ff -> fi) = { tt }
  (ff -> uu) = { tt }
  (ff -> ii) = { tt }
  (ff -> it) = { tt }
  (ff -> tt) = { tt }
  (fi -> ff) = { uu ii it tt }
  (fi -> fi) = { tt }
  (fi -> uu) = { tt }
  (fi -> ii) = { tt }
  (fi -> it) = { tt }
  (fi -> tt) = { tt }
  (uu -> ff) = { fi ii it tt }
  (uu -> fi) = { fi ii it tt }
  (uu -> uu) = { tt }
  (uu -> ii) = { fi ii it tt }
  (uu -> it) = { tt }
  (uu -> tt) = { tt }
  (ii -> ff) = { fi uu it tt }
  (ii -> fi) = { fi uu it tt }
  (ii -> uu) = { fi uu it tt }
  (ii -> ii) = { tt }
  (ii -> it) = { tt }
  (ii -> tt) = { tt }
  (it -> ff) = { fi uu ii tt }
  (it -> fi) = { fi uu ii tt }
  (it -> uu) = { fi uu ii tt }
  (it -> ii) = { fi uu ii tt }
  (it -> it) = { tt }
  (it -> tt) = { tt }
  (tt -> ff) = { ff }
  (tt -> fi) = { fi }
  (tt -> uu) = { uu }
  (tt -> ii) = { ii }
  (tt -> it) = { it }
  (tt -> tt) = { tt }
  Please either press [Enter] to continue or [Ctrl-C] to make some changes:
  The search will possibly start from #0000000000000000 (base 6),
  but there may be improvements here.

  From the .aux file: 0 current solutions;
  The current best score: 0
  The current minimal score as a filter: 0

  In the table, I have dared fill in 4 entries for you.
  The new implication table:
     | ff fi uu ii it tt
  ---+------------------
  ff | tt tt tt tt tt tt
  fi | .. tt tt .. tt tt
  uu | .. .. tt .. tt tt
  ii | .. .. .. tt tt tt
  it | .. .. .. .. tt tt
  tt | ff fi uu ii it tt

  Having removed the singletons, there are 12 digits left.
  More precisely, in the base 6, the search will actually start from #251111111111
  Your minimum score to be used as a filter (a non-negative number): 0
  Searching...
```

```
Solution #1
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | tt tt tt tt tt tt
uu | tt tt tt tt tt tt
ii | tt tt tt tt tt tt
it | tt tt tt tt tt tt
tt | ff fi uu ii it tt
Score = 51
```

Without checking neither contrapositive property nor $\big((a \to b) \to a\big) \to a$, the run outputs 5 solutions (the personal best score is 93, but its evaluation can be changed):

```
Solution #1
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | tt tt tt tt tt tt
uu | tt tt tt tt tt tt
ii | tt tt tt tt tt tt
it | ff ff ff ff tt tt
tt | ff fi uu ii it tt
Score = 71

Solution #2
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | tt tt tt tt tt tt
uu | ff ff tt ff tt tt
ii | tt tt tt tt tt tt
it | ff ff uu ff tt tt
tt | ff fi uu ii it tt
Score = 84

Solution #3
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | ff tt tt tt tt tt
uu | ff fi tt fi tt tt
ii | ff tt tt tt tt tt
it | ff fi uu fi tt tt
tt | ff fi uu ii it tt
Score = 90

Solution #4
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | tt tt tt tt tt tt
uu | ff ff tt ii tt tt
ii | ff ff uu tt tt tt
it | ff ff uu ii tt tt
tt | ff fi uu ii it tt
Score = 91

Solution #5
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | ff tt tt tt tt tt
uu | ff fi tt ii tt tt
ii | ff fi uu tt tt tt
it | ff fi uu ii tt tt
tt | ff fi uu ii it tt
Score = 93
```

B. Checking with an uncertainty equivalence, i.e. $(ff \leftrightarrow fi) = tt, (it \leftrightarrow tt) = tt$*:*

Checking the properties, the run found two impossibilities: $(uu \to ff) = \{\}, (ii \to ff) = \{\}$. No need for searching;

Not checking $\big((a \to b) \to a\big) \to a$, no solution after searching;

Not checking contrapositive, 8 solutions. (the subjective 'best score' is 98 at the solution #8):

```
Solution #1
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | it tt it it it tt
uu | it it tt it it tt
ii | it it it tt it tt
it | ff ff ff ff tt tt
tt | ff fi uu ii it tt
Score = 83

Solution #2
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | it tt it it it tt
uu | it ff tt ff it tt
ii | it ff ff tt it tt
it | it ff ff ff tt tt
tt | ff fi uu ii it tt
Score = 95

Solution #3
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | it tt it it it tt
uu | ff ff tt ff it tt
ii | it uu uu tt it tt
it | ff ff it ff tt tt
tt | ff fi uu ii it tt
Score = 95
```

```
Solution #4

-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | ff tt it it it tt
uu | ff it tt fi it tt
ii | ff it fi tt it tt
it | ff it fi fi tt tt
tt | ff fi uu ii it tt
Score = 95

Solution #5
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | ff tt it it it tt
uu | ff fi tt fi it tt
ii | ff it it tt it tt
it | ff fi uu fi tt tt
tt | ff fi uu ii it tt
Score = 97

Solution #6
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | ff tt it it it tt
uu | ff fi tt fi it tt
ii | ff it uu tt it tt
it | ff fi it fi tt tt
tt | ff fi uu ii it tt
Score = 97

Solution #7
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | it tt it it it tt
uu | ff ff tt ii it tt
ii | ff ff uu tt it tt
it | ff ff uu ii tt tt
tt | ff fi uu ii it tt
Score = 97

Solution #8
-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | ff tt it it it tt
uu | ff fi tt ii it tt
ii | ff fi uu tt it tt
it | ff fi uu ii tt tt
tt | ff fi uu ii it tt
Score = 98
```

C. Checking with another uncertainty equivalence $(a = a, ff \leftrightarrow fi = it, tt \leftrightarrow it = it, ff \; otherwise)$*:*

Without checking ((a→b)→a)→a, no solutions after searching;

Without checking the contrapositive, 1 solution:

```
Solution #1

-> | ff fi uu ii it tt
---+------------------
ff | tt tt tt tt tt tt
fi | tt tt tt tt tt tt
uu | tt tt tt tt tt tt
ii | tt tt tt tt tt tt
it | tt tt tt tt tt tt
tt | ff fi uu ii it tt
Score = 51
```

D. Checking with a weaker equivalence $(a = a, ff \leftrightarrow fi = it, tt \leftrightarrow it = it, fi \leftrightarrow ii = it, ii \leftrightarrow it = it, ff \; otherwise)$*:*

Checking the properties, no solution, no searching;

Without checking the contrapositive, 1 solution with the same table as above.

III. THE CODE IN C

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<time.h>
#include<sys/time.h>

#define FALSE 0
#define TRUE 1

// Number of values
#define NN 6

// True
#define TT 5

#define UU 2
#define IT 4

#define M NN-1

#define datname "amvlog.dat"
#define auxname "amvlog.aux"
#define logname "amvlog.log"

FILE *aux;

typedef char word[3];

const word v[NN+1] = {"ff", "fi", "uu", "ii", "it",
"tt", ".."};

typedef struct {
  float min, false, true, max;
} typsamplevalues;

typsamplevalues sv = { -1.0, -0.33, 0.33,  1.0 }; //
```

```
default


  typsamplevalues sa[NN] = { { -1.0, -0.33, 0.33, -0.5
}, // ff
                           { -1.0, -0.33, 0.33,  0.0
}, // fi
                           { -1.0, -0.33, 0.33,  1.0
}, // uu
                           { -0.2, -0.33, 0.33,  0.2
}, // ii
                           { -0.2, -0.33, 0.33,  1.0
}, // it
                           {  0.5, -0.33, 0.33,  1.0 }
// tt
                          };

  typedef int tab[NN][NN];

  typedef struct {
    int i, j;
  } indices;

  typedef char typconstr[NN+1];

  typconstr constr[NN][NN];

  char cons[NN][NN][NN];

  indices hash[NN*NN];

  tab and, or, imp, eqv, impmsk;


  int not[NN];

  FILE *in;

  // nn is the last index, i.e. the index of the
  // last digit in the base NN
  int nn = -1, nsol = 0;

  int minscore = 0, bestscore = 0;

  char findindex(word w) {
    char k =0, found = FALSE;
    while (!found && k<=M+1)
      if (strcmp(w,v[k]) == 0) found = TRUE; else k++;
    return k;
  }

  void lerimp(tab t) {
    word w;
    char found;
    int i, j, k;

    for (i=0; i<=M; i++) {
      for (j=0; j<=M; j++) {
        fscanf(in,"%s ",w);
        found = FALSE;
        for (k = 0; !found && k<=M+1; k++) {
          if (strcmp(w,v[k])==0) {
            found = TRUE;
            t[i][j] = k;
            if (k == NN) {
              hash[++nn].i = i; hash[nn].j = j;
              imp[i][j] = 0;
            }
          }
        }
        if (!found) {
          printf("Error in the implication table:
incorrect data in the input .dat file: %s.\n",w);
          exit(1);
        }
      }
    }
  }

  void ler(tab t) {
    word w;
    char found;
    int i, j, k;

    for (i=0; i<=M; i++) {
      for (j=0; j<=M; j++) {
        fscanf(in,"%s ",w);
        found = FALSE;
        for (k = 0; !found && k<=M+1; k++) {
          if (strcmp(w,v[k])==0) {
            found = TRUE;
            t[i][j] = k;
            // if (k == NN) { /* Nothing */ }
          }
        }
        if (!found) {
          printf("Error: incorrect data in the input
.dat file: %s.\n",w);
          exit(2);
        }
      }
    }
  }

  void myfscanf(FILE *in) {
    char c;
    printf("The remaining characters of the line
regarded as comment: ");
    do {
      c = fgetc(in);
      printf("%c",c);
    } while (c != '\n' && !feof(in));
  }

  void escreve(tab t) {
    int i,j;
    for (i=0; i<=M; i++) {
      printf("%s |",v[i]);
      for (j=0; j<=M; j++)
        printf(" %s",v[t[i][j]]);
      printf("\n");
    }
  }

  void grava(tab t) {
    FILE *log = fopen(logname,"a");
    int i,j, lscore = 0;
    fprintf(log,"\n");
    fprintf(log,"Solution #%d\n",nsol);
    fprintf(log,"-> | ");
    for (i=0; i<=M; i++) fprintf(log,"%s ",v[i]);
    fprintf(log,"\n---+");
    for (i=0; i<=M; i++) fprintf(log,"---");
    fprintf(log,"\n");
    for (i=0; i<=M; i++) {
      fprintf(log,"%s |",v[i]);
      for (j=0; j<=M; j++) {
        lscore += NN - t[i][j];
        fprintf(log," %s",v[t[i][j]]);
      }
```

```
      fprintf(log,"\n");
    }
    fprintf(log,"%d\n",lscore);
    if (lscore > bestscore)
      bestscore = lscore;
    fclose(log);
  }


  unsigned long myexp(int base, int x) { // It is just
to provide some idea in the base 10.
    unsigned long n = 1;                  // It does not
help to get any result in this program,
    while (x > 0) {                       // but instead
to give the user the size of the search.
      n = n * base;                       // If there is
any overflow, just remove its call.
      x--;
    }
    return n;
  }

  void err(char *e, char n, char a, char b) {
    (*e)++;
    printf("Erro #%d em %s -> %s\n",n,v[a],v[b]);
  }

  char elementmask(tab imp, char a, char b, char c) {
    char erro = 0, c2 = imp[a][b];

    imp[a][b] = c;

    // Contrapositive
    if (a != NN && b != NN && imp[a][b] != NN &&
imp[not[b]][not[a]] != NN &&
        eqv[TT][eqv[imp[a][b]][imp[not[b]][not[a]]]]   !=
TT)
      erro++;

        // a and b --> a or b
        if (imp[and[a][b]][or[a][b]] != NN &&
            eqv[TT][imp[and[a][b]][or[a][b]]]   !=   TT)
erro++;

        // P6. a => (b => a)
        if (a != NN && b != NN && imp[b][a] != NN &&
            imp[a][imp[b][a]] != NN &&
            eqv[TT][imp[a][imp[b][a]]] != TT) erro++;

        // P7. ((a => b) => a) => a
        if (a != NN && b != NN && imp[a][b] != NN &&
            imp[imp[a][b]][a] != NN &&
            imp[imp[imp[a][b]][a]][a] != NN &&
            eqv[imp[imp[imp[a][b]][a]][a]][TT]   !=   TT)
erro++;

        c = 0;
        while (erro == 0 && c<=M) {
          // P2. (a => (b => c)) => (b => (a => c))
          if (a != NN && b != NN && c != NN && imp[b][c]
!= NN &&
              imp[a][imp[b][c]] != NN && imp[a][c] != NN
&&
              imp[b][imp[a][c]] != NN &&
              imp[imp[a][imp[b][c]]][imp[b][imp[a][c]]]
!= NN &&

eqv[TT][imp[imp[a][imp[b][c]]][imp[b][imp[a][c]]]] != TT)
                erro++;
          // P3. (c => a) => ((b => c) => (b => a))
          if (c != NN && a != NN && imp[c][a] != NN &&
              b != NN && imp[b][c] != NN && imp[b][a] !=
NN &&
              imp[imp[b][c]][imp[b][a]] != NN &&
              imp[imp[c][a]][imp[imp[b][c]][imp[b][a]]]
!= NN &&

eqv[TT][imp[imp[c][a]][imp[imp[b][c]][imp[b][a]]]] != TT)
                erro++;
          // P4. (c => a) => ((a => b) => (c => b))
          if (c != NN && a != NN && imp[c][a] != NN &&
              imp[a][b] != NN &&
              imp[c][b]         !=         NN         &&
imp[imp[a][b]][imp[c][b]] != NN &&
              imp[imp[c][a]][imp[imp[a][b]][imp[c][b]]]
!= NN &&

eqv[TT][imp[imp[c][a]][imp[imp[a][b]][imp[c][b]]]] != TT)
                erro++;
          // P5. (a => (b => c)) => ((a => b) => (a =>
c))
  // Without testing, just place a comment
          if (a != NN && b != NN && c != NN && imp[b][c]
!= NN &&
              imp[a][imp[b][c]] != NN &&
              imp[a][b] != NN &&
              imp[a][c]         !=         NN         &&
imp[imp[a][b]][imp[a][c]] != NN &&

imp[imp[a][imp[b][c]]][imp[imp[a][b]][imp[a][c]]]
              != NN && eqv[TT][
      imp[imp[a][imp[b][c]]][imp[imp[a][b]][imp[a][c]]]
              ] != TT) erro++;
          c++;
        }
    imp[a][b] = c2;
    return erro;
  }

  char * myconstr(typconstr c) {
    char * d = malloc(NN);
    char i = 0;

    while (c[i] != '\0') {
      d[i] = c[i] + '0' - 1;
      i++;
    }
    d[i] = '\0';
    return d;
  }

  char mask(tab imp) {
    char a, b, c;
    char erro = 0;

    a = 0;
    while (erro == 0 && a<=M) {
      // P1. Identity:
      if (imp[a][a] != NN && eqv[imp[a][a]][TT] != TT)
err(&erro,2,a,b);
      b=0;
      while (erro == 0 && b<=M) {
        constr[a][b][0] = '\0';
        // Contrapositive
        if (imp[a][b] != NN && imp[not[b]][not[a]] != NN
&&
        eqv[TT][eqv[imp[a][b]][imp[not[b]][not[a]]]]   !=
TT) err(&erro,3,a,b);
        // a and b --> a or b
        if (imp[and[a][b]][or[a][b]] != NN &&
```

```
            eqv[TT][imp[and[a][b]][or[a][b]]]    !=    TT)
err(&erro,4,a,b);
        // P6. a => (b => a)
        if (a != NN && b != NN && imp[b][a] != NN &&
            imp[a][imp[b][a]] != NN &&
            eqv[TT][imp[a][imp[b][a]]]       !=       TT)
err(&erro,7,a,b);
        // P7. ((a => b) => a) => a
        if (a != NN && b != NN && imp[a][b] != NN &&
            imp[imp[a][b]][a] != NN &&
            imp[imp[imp[a][b]][a]][a] != NN &&
            eqv[imp[imp[imp[a][b]][a]][a]][TT]   !=   TT)
err(&erro,8,a,b);
        c = 0;
        while (erro == 0 && c<=M) {
          // P2. (a => (b => c)) => (b => (a => c))
          if (a != NN && b != NN && c != NN && imp[b][c]
!= NN &&
              imp[a][imp[b][c]] != NN && imp[a][c] != NN
&&
              imp[b][imp[a][c]] != NN &&
              imp[imp[a][imp[b][c]]][imp[b][imp[a][c]]]
!= NN &&

eqv[TT][imp[imp[a][imp[b][c]]][imp[b][imp[a][c]]]] != TT)
                err(&erro,9,a,b);
          // P3. (c => a) => ((b => c) => (b => a))
          if (c != NN && a != NN && imp[c][a] != NN &&
              b != NN && imp[b][c] != NN && imp[b][a] !=
NN &&
              imp[imp[b][c]][imp[b][a]] != NN &&
              imp[imp[c][a]][imp[imp[b][c]][imp[b][a]]]
!= NN &&

eqv[TT][imp[imp[c][a]][imp[imp[b][c]][imp[b][a]]]] != TT)
                err(&erro,10,a,b);
          // P4. (c => a) => ((a => b) => (c => b))
          if (c != NN && a != NN && imp[c][a] != NN &&
              imp[a][b] != NN &&
              imp[c][b]         !=         NN         &&
imp[imp[a][b]][imp[c][b]] != NN &&
              imp[imp[c][a]][imp[imp[a][b]][imp[c][b]]]
!= NN &&

eqv[TT][imp[imp[c][a]][imp[imp[a][b]][imp[c][b]]]] != TT)
                err(&erro,11,a,b);
          // P5. (a => (b => c)) => ((a => b) => (a =>
c))
          if (a != NN && b != NN && c != NN && imp[b][c]
!= NN &&
              imp[a][imp[b][c]] != NN &&
              imp[a][b] != NN &&
              imp[a][c]         !=         NN         &&
imp[imp[a][b]][imp[a][c]] != NN &&

imp[imp[a][imp[b][c]]][imp[imp[a][b]][imp[a][c]]]
              != NN && eqv[TT][

imp[imp[a][imp[b][c]]][imp[imp[a][b]][imp[a][c]]]
              ] != TT) err(&erro,12,a,b);
            c++;
          }
          b++;
        }
        a++;
      }
      return erro == 0;
    }

    void fillconstr(tab imp) {
      char a = 0, b = 0, dotdot = FALSE;

      for (a = 0; a < NN; a++) {
        for (b = 0; b < NN; b++) {
          if (imp[a][b] != NN) {
            char len = strlen(constr[a][b]);

            constr[a][b][len] = imp[a][b] + 1;   // + 1 is
for avoiding the null char
            constr[a][b][len+1] = '\0';
            cons[a][b][imp[a][b]] = 1;
          }
          else dotdot = TRUE;
        }
      }
      if (dotdot) {
        for (a = 0; a < NN; a++) {
          for (b = 0; b < NN; b++) {
            if (imp[a][b] == NN) {
              char c = 0;
              constr[a][b][c] = '\0';
              while (c < NN) {
                if      (imp[a][b]      ==      NN      &&
elementmask(imp,a,b,c) == 0) {
                  char len = strlen(constr[a][b]);
                  constr[a][b][len] = c + 1;    // + 1 is
for avoiding the null char
                  constr[a][b][len+1] = '\0';
                  cons[a][b][c] = 1;
                }
                else if (c < NN) cons[a][b][c] = 0;
                c++;
              }
            }
          }
        }
      }
    }

    char inconstr(char v, char a, char b) {
      char found = FALSE;
      char k = 0;
      v++;
      while (!found && constr[a][b][k] != '\0') {
        if (constr[a][b][k] == v) found = TRUE;
        k++;
      }
      return found;
    }

    void insertconstr(char v, char a, char b) {
      char k = 0, m;
      char found = FALSE;
      char len = strlen(constr[a][b]);

      while (constr[a][b][k] < v) k++;
      if (constr[a][b][k] > v) {
        constr[a][b][len+1] = '\0';
        for (m = len; m > k; m--) constr[a][b][m] =
constr[a][b][m-1];
        constr[a][b][m] = v+1;     // +1 is for avoiding
the null char
      }
    }

    char delconstr(char v, char a, char b) {
      char found = FALSE;
      char k = 0;
      v++;
      while (!found && constr[a][b][k] != '\0')
```

```
    if (constr[a][b][k] == v) found = TRUE; else k++;
    if (found) {
      do {
        constr[a][b][k] = constr[a][b][k+1];
        k++;
      } while (constr[a][b][k] != '\0');
    }
    return found;
  }

  void printaconstr(char a, char b) {
    char c = 0;
    printf("(%s -> %s) = {",v[a],v[b]);
    while (constr[a][b][c] != '\0') {
      printf(" %s",v[constr[a][b][c]-1]);
      c++;
    }
    printf(" }\n");
  }

  void pause(void) { char c; scanf("%c",&c); }
  void printconstr(void) {
    char a = 0, b = 0;
    for (a = 0; a < NN; a++)
      for (b = 0; b < NN; b++) printaconstr(a,b);
  /*
    printf("Please press [Enter] to see their direct
access structure: "); pause();
    printf("\nDirect access:\n");
    for (a = 0; a < NN; a++) {
      for (b = 0; b < NN; b++) {
        char c = 0;
        printf("%s -> %s = [",v[a],v[b]);
        while (c < NN) {
          printf(" %d",cons[a][b][c]);
          c++;
        }
        printf(" ]\n");
      }
    }
  */
  }
  int readrules(FILE *in) {
    word tail, head, result;
    char w[10];
    char t, h, r, positive, k, error = FALSE, end =
FALSE, nrules = 0;
    fscanf(in,"%s ",w);
    if (strcmp(w,"rules") == 0) {
      printf("Any rules?\n");
    }
    do {
      fscanf(in,"%s ",w);
      if (strcmp(w,"end") == 0 || feof(in)) end = TRUE;
      else {
        char c;
        if (strcmp(w,"//") == 0) // if a full comment
line
          myfscanf(in);
        else {
          if (strcmp(w,"!!") == 0) {
            positive = FALSE;
            fscanf(in,"%s ",tail);  // read the tail of
a rule
          } else {
            positive = TRUE;
            strcpy(tail,w);             // the tail had
already been read
          }
          t = findindex(tail);
          fscanf(in,"%s ",w);      // ->
          fscanf(in,"%s ",head);    // read the head of
the same rule
          h = findindex(head);
          fscanf(in,"%s ",w);      // ==
          fscanf(in,"%s",result);  // read the result of
the same rule
          nrules++;
          r = findindex(result);
          printf("  %s ((%s -> %s) == %s)\n",((positive)
? "positive" : "negative"),tail,head,result);

          myfscanf(in);                 // the remaining
chars of the line are ignored
          if (positive) {
            if (inconstr(r,t,h)) {
              printf("  Warning: redundance in %s -> %s
resulting in %s.\n",tail,head,result);
              printaconstr(t,h);
            }
            else {
              if    (strlen(constr[t][h])    >    1    &&
impmsk[t][h] == NN) {
                printf("\nError in the rule %s -> %s for
the result %s will not be found.\n",tail,head,result);
                printf("Check     the     incompatibility
between the rule and the following possibilities:\n");
                printaconstr(t,h);
                exit(3);
              }
              else {
                printf("  Warning: I included the above
rule to the table-mask element:\n");
                insertconstr(r,t,h);
                printaconstr(t,h);
              }
            }
            cons[t][h][r] = 1;
          }
          else { // not positive
            if (inconstr(r,t,h)) {
              delconstr(r,t,h);
              printf("The implication %s -> %s resulting
in     %s     has     been     excluded     from     the
considerations.\n",tail,head,result);
              printf("Updated constraint: ");
              printaconstr(t,h);
            }
            else {
              printf("Warning: negative rule %s -> %s ==
%s unnecessary:\n",tail,head,result);
              printf("Previous consideration: ");
              printaconstr(t,h);
            }
            cons[t][h][r] = 0;
          }
        }
      }
    } while (!end && !feof(in));
    if (nrules == 0) printf("That is ok, I have found no
rule in the input data.\n"); else if (nrules == 1)
    printf("I  have  found  a  single  rule.\n");  else
printf("%d rules were detected.\n",nrules);
    error = FALSE;
    for (t = 0; t < NN; t++)
      for (h = 0; h < NN; h++)
        if (constr[t][h][0] == '\0') {
          if (!error) {
            error = TRUE;
            printf("Error: I am affraid, there are one
```

```
or more impossibilities. Check the following:\n");
          }
          printaconstr(t,h);
        }
    if (error) {
      printf("The whole set of constraints:\n");
      printconstr();
      printf("Thus, no need for searching.\n");
      exit(4);
    }
  }
char tabok(tab imp) {
    int a, b, c;
    char erro = 0;

    a = 0;

    while (erro == 0 && a<=M) {

      // P1. Identity:
      if (imp[a][a] != TT) erro++; else {

        b=0;

        while (erro == 0 && b<=M) {

          if (cons[a][b][imp[a][b]] > 0) {

            // Contrapositive

            if  (eqv[imp[a][b]][imp[not[b]][not[a]]]  !=
TT) erro++; else

            // a and b --> a or b
            if (eqv[TT][imp[and[a][b]][or[a][b]]] != TT)
erro++; else
            // P6. a => (b => a)
            if   (eqv[TT][imp[a][imp[b][a]]]   !=   TT)
erro++; else
            // P7. ((a => b) => a) => a
            if  (eqv[TT][imp[imp[imp[a][b]][a]][a]]  !=
TT) erro++; else
  {
              c = 0;
              while (erro == 0 && c<=M) {
                // P2. (a => (b => c)) => (b => (a =>
c))
                if
(eqv[TT][imp[imp[a][imp[b][c]]][imp[b][imp[a][c]]]]   !=
TT) erro++; else

                // P3. (c => a) => ((b => c) => (b =>
a))
                if
(eqv[TT][imp[imp[c][a]][imp[imp[b][c]][imp[b][a]]]]   !=
TT) erro++; else

                // P4. (c => a) => ((a => b) => (c =>
b))
                if
(eqv[TT][imp[imp[c][a]][imp[imp[a][b]][imp[c][b]]]]   !=
TT) erro++; else

                // P5. (a => (b => c)) => ((a => b) =>
(a => c))
                // Distributive
                if
(eqv[TT][imp[imp[a][imp[b][c]]][imp[imp[a][b]][imp[a][c]]
]] != TT) erro++; else
                c++;
              } // while
            } // end else
          } // end if
          b++;
        } // while
      } // end else

      a++;
    }
    return erro == 0;
  }

  void displaynumber(void) {
    for (char kk = 0; kk <= nn; kk++)
      printf("%c",'0'+imp[hash[kk].i][hash[kk].j]);
    printf("\n");
  }

  void removesingletons(void) {
    char nst = 0;
    char x = 0;
    while (x <= nn) {
      if (constr[hash[x].i][hash[x].j][1] == '\0') {
        nst++;
        impmsk[hash[x].i][hash[x].j]                    =
constr[hash[x].i][hash[x].j][0]-1;
        for  (char  y=x+1;  y  <=  nn;  y++)  hash[y-1]  =
hash[y];
        nn--;
      }
      x++;
    }

    if (nst > 0) {
      printf("In the table, I have dared fill in ");
      if (nst == 1) printf("an entry for you.\n");
      else printf("%d entries for you.\n",nst);
      printf("The new implication table:\n");
      printf("   | ");
      for (char i=0; i<=M; i++) printf("%s ",v[i]);
      printf("\n---+");  for  (char  i=0;  i<=M;  i++)
printf("---");
      printf("\n");
      escreve(impmsk);

      printf("\n");
      printf("Having  removed  the  singletons,  there  are
%d digits left.\n",nn+1);
      printf("More precisely, in the base %d, the search
will actually start from #",NN);
      displaynumber();
    }
  }

  int thescore(tab t) {
    char a, b;
    int lscore = 0;

    for (a = 0; a < NN; a++)
      for (b = 0; b < NN; b++)
        lscore += NN - t[a][b];
    return lscore;
  }
  /* That is for a large number of values
  int sorteios(void) {
    int k = nn, m;

    while (k >= 0 && imp[hash[k].i][hash[k].j] == 0) k--
;
    m = k + 1;
```

```
    if (m > 0) {
      for (int i = 1; i <= 10; i++) {
        k = m;
        while (k <= nn) {
          imp[hash[k].i][hash[k].j] = rand() % NN;
          k++;
        }
        if (tabok(imp)) {
          printf("Solution #%d (randomic):\n",++nsol);
          escreve(imp);
          printf("Score = %d\n\n",thescore(imp));
          grava(imp);
        }
      }
      for     (k     =     m;     k     <=     nn;     k++)
imp[hash[k].i][hash[k].j] = 0;
    }
    else printf("Programming mistake.\n");
  }
  */
  char firstconstr(char x) {
    return constr[hash[x].i][hash[x].j][0] - 1;
  }
  char lastconstr(char x) {
    return
constr[hash[x].i][hash[x].j][strlen(constr[hash[x].i][has
h[x].j])-1] - 1;
  }
  void checkanddo(int x) {
    time_t t; struct tm *tv;
    char k;
    time(&t);
    tv = localtime(&t);
    if (tv->tm_min % 5 == 0 && tv->tm_sec == 0) {
      FILE *aux = fopen(auxname,"wt");
      for (k = 0; k <= nn; k++)
        fprintf(aux,"%c",'0'+imp[hash[k].i][hash[k].j]);
      fprintf(aux," %d %d\n\n",nsol,bestscore);
      for (char i=0; i<=M; i++) {
        for (char j=0; j<=M; j++) {
          fprintf(aux,"%s",v[imp[i][j]]);
          if    (j    ==    M)    fprintf(aux,"\n");    else
fprintf(aux," ");
        }
      }
      fclose(aux);
      while (tv->tm_min % 5 == 0 && tv->tm_sec == 0) {
        time(&t);
        tv = localtime(&t);
      }
    }
    if            (thescore(imp)>=bestscore            &&
cons[hash[x].i][hash[x].j][imp[hash[x].i][hash[x].j]]  !=
0 && tabok(imp)) {
      printf("Solution #%d\n",++nsol);
      printf("-> | ");
      for (char i=0; i<=M; i++) printf("%s ",v[i]);
      printf("\n---+");
      for (char i=0; i<=M; i++) printf("---");
      printf("\n");
      escreve(imp);
      printf("Score = %d\n\n",thescore(imp));
      grava(imp);
    }
  }
void initnumber(tab imp) {
    for     (char     x=0;     x     <=     nn;     x++)
imp[hash[x].i][hash[x].j] = firstconstr(x);
  }
```

```
  int main(void) {
    FILE *aux;
    FILE *log = fopen(logname,"a");
    fprintf(log,"A new run:\n");
    fclose(log);
    printf("Welcome to a run searching for a %d-valued
logic.\n",NN);
  //  srand(time(NULL));
    in=fopen(datname,"rt");
    if (in==NULL) {
      printf("Could  not  open  the  file  %s\n",datname);
exit(5);
    }
    else {
      char i, j, k;
      word w;
      char c, found, end;
      char firsttime = TRUE;
      char yn = 'n';
      for (j=0; j<=M; j++) {
        fscanf(in,"%s ",w);
        found = FALSE;
        for (k=0; !found && k<=M; k++) {
          if (strcmp(v[k],w)==0) {
            found = TRUE;
            not[j] = k;
          }
        }
        if (!found) {
          printf("Incorrect  data  in  the  input  file:
%s.\n",w);
          exit(6);
        }
      }
      // Check double negation
      for (j=0; j<=M; j++) {
        if (j != not[not[j]]) {
          printf("Error:  not  not  %s  does  not  result  in
%s but instead in %s.\n",v[j],v[j],v[not[not[j]]]);
          exit(7);
        }
      }
      ler(and);
      for (i=0; i<=M; i++) {
        if (and[i][i] == NN) and[i][i] = i; else
        if (and[i][i] != i) {
          printf("Error:  Sorry,  in  this  logic,  a  and  a
has to result in a.\n");
          exit(8);
        }
        for (j=0; j<=M; j++) {
          if (and[i][j] == NN) {
            if (and[j][i] != NN) and[i][j] = and[j][i];
            else {
              if (firsttime) {
                printf("Warning:   Sorry,   the   current
version does not permit search over ");
                printf("conjunctions.  In  the  present
case, (%s and %s) has no value.\n",v[i],v[j]);
                printf("Would  you  like  my  suggestions
for such open values (y/n)? ");
                scanf("%c",&yn);
                { char c; scanf("%c",&c); } // line feed
                firsttime = FALSE;
              }
              if (yn == 'y') {
  #define min(x,y) ((x)<(y))?(x):(y)
                typsamplevalues lsv = sv;
                lsv.min = min(sa[i].min,sa[j].min);
                lsv.max = min(sa[i].max,sa[j].max);
```

```
                if (lsv.max < sv.false) and[i][j] = 0;
else
                if (lsv.min < sv.false && lsv.max <
sv.true) and[i][j] = 1; else
                if (lsv.min < sv.false && lsv.max >=
sv.true) and[i][j] = 2; else
                if (lsv.min >= sv.false && lsv.max <
sv.true) and[i][j] = 3; else
                if (lsv.min >= sv.false && lsv.max >=
sv.true) and[i][j] = 4; else
                if (lsv.min >= sv.true && lsv.max >=
sv.true) and[i][j] = 5;
                else {
                  printf("Sorry: a programming bug.
Please input a full conjunction table and rerun.\n");
                  exit(9);
                }
                printf("Suggested value: %s and %s =
%s\n",v[i],v[j],v[and[i][j]]);
                and[j][i] = and[i][j];
              }
              else {
                printf("Please complete your conjunction
table.\n");
                exit(10);
              }
            }
          }
          else {
            if (and[j][i] != NN && and[i][j] !=
and[j][i]) {
              printf("Error: the commutative property on
(%s and %s) does not hold.",v[i],v[j]);
              exit(11);
            }
          }
        }
      }
      for (i=0; i<=M; i++) {
        or[i][i] = i;
        for (j=0; j<=i; j++) {
          or[j][i]         =         or[i][j]         =
not[and[not[i]][not[j]]]; // De Morgan 1
        }
      }
      for (i=0; i<=M; i++) {
        for (j=0; j<=i; j++) {
          if (not[and[i][j]] != or[not[i]][not[j]]){
            printf("Error: De Morgan 2, not(%s and %s)
<-> not %s or not %s, does not hold.\n",
              v[i],v[j],v[i],v[j]);
            printf("not(%s and %s) = %s whereas not %s
or not %s = %s.\n",

v[i],v[j],not[and[i][j]],v[i],v[j],or[not[i]][not[j]]);
            exit(12);
          }
        }
      }
      char erro = 0;
      for (char a=0; erro == 0 && a<=M; a++) {
        for (char b=0; erro == 0 && b<=M; b++) {
          for (char c=0; erro == 0 && c<=M; c++) {
                // associative and
                if          (and[and[a][b]][c]          !=
and[a][and[b][c]])
                  { erro++;
                    printf("Associative and does not
hold: (%s and %s) and %c.\n",a,b,c); }
                // associative or
                if (or[or[a][b]][c] != or[a][or[b][c]])
                  { erro++;
                    printf("Associative or does not
hold: (%s or %s) or %c.\n",a,b,c); }
                // distributive and-or
                if          (or[and[a][b]][c]          !=
and[or[a][c]][or[b][c]])
                  { erro++;
                    printf("Distributive and-or does not
hold: (%s and %s) or %c.\n",a,b,c); }
                // distributive or-and
                if          (and[or[a][b]][c]          !=
or[and[a][c]][and[b][c]])
                  { erro++;
                    printf("Distributive or-and does not
hold: (%s or %s) and %c.\n",a,b,c); }
          }
        }
      }
      if (erro > 0) {
        if (erro == 1) printf("There is one fatal
mistake in the and-or tables.\n");
        else printf("There are %d fatal mistakes in the
and-or tables.\n");
        exit(13);
      }
      ler(eqv);
      lerimp(impmsk);
      printf("\nNegation:\n");
      printf("   | ");
      for (i=0; i<=M; i++) printf("%s ",v[i]);
      printf("\n---+"); for (i=0; i<=M; i++) printf("---
"); printf("\n");
      printf("   | ");
      for (j=0; j<=M; j++) printf("%s ",v[not[j]]);
      printf("\n");
      printf("\nConjunction:\n");
      printf("   | ");
      for (i=0; i<=M; i++) printf("%s ",v[i]);
      printf("\n---+"); for (i=0; i<=M; i++) printf("---
"); printf("\n");
      escreve(and);
      printf("\nDisjunction:\n");
      printf("   | ");
      for (i=0; i<=M; i++) printf("%s ",v[i]);
      printf("\n---+"); for (i=0; i<=M; i++) printf("---
"); printf("\n");
      escreve(or);
      printf("\nEquivalence:\n");
      printf("   | ");
      for (i=0; i<=M; i++) printf("%s ",v[i]);
      printf("\n---+"); for (i=0; i<=M; i++) printf("---
"); printf("\n");
      escreve(eqv);
      for (i = 0; i <= M; i++) {
        for (j = 0; j <= M; j++)
          imp[i][j] = impmsk[i][j];
      }
      printf("\nImplication:\n");
      printf("   | ");
      for (i=0; i<=M; i++) printf("%s ",v[i]);
      printf("\n---+"); for (i=0; i<=M; i++) printf("---
"); printf("\n");
      escreve(imp);
      printf("\n");
      /*
         Make sure that the .dat file ends with a
         line with the word 'end'. The remaining
         lines will be ignored and can be used as
         comment, but the word end is necessary for
```

```
        there is no parser in this short program.
      */
      if (!mask(imp)) {
        printf("The constraints make any solution
impossible.\n");
        exit(14);
      }
      else {
        char ch;
        fillconstr(imp);
        readrules(in); fclose(in);
        printf("Please press [Enter] to see the set of
constraints: ");
        scanf("%c",&ch);
        printconstr();
        printf("Please either press [Enter] to continue
or [Ctrl-C] to make some changes: ");
        scanf("%c",&ch);
        aux = fopen(auxname,"rt");
        if (aux == NULL)
          printf("Could not open the file
%s.\n",auxname);
        else {
          int x = 0;
          time_t t; struct tm *tv;
          printf("The search will possibly start from
#");
          k = 0;
          do {
            do fscanf(aux,"%c",&c); while ((c == 10)
                && !feof(aux)); // possible line feeds
            if (c >= '0' && c <= '9')
imp[hash[k].i][hash[k].j] = c - '0';
            k++;
            if (c >= '0' && c <= '9') printf("%c",c);
          } while (c >= '0' && c <= '9');
          printf(" (base %d),\nbut there may be
improvements here.\n",NN);
          fscanf(aux,"%d",&nsol);
          printf("\nFrom the .aux file: %d current
solutions;\n",nsol);
          fscanf(aux,"%d",&bestscore);
          printf("The current best score:
%d\n",bestscore);
          fscanf(aux,"%d",&minscore);
          printf("The current minimal score as a filter:
%d\n",minscore);
          fclose(aux);
          printf("\n");
          initnumber(imp);
          removesingletons();
          checkanddo(x);
          printf("Your minimum score to be used as a
filter (a non-negative number): ");
          scanf("%d",&minscore);
          printf("Searching...\n");
          end = FALSE;
          x = 0;
          while (!end && x <= nn) {
            imp[hash[x].i][hash[x].j]++;
            while (!end && x <= nn+1 &&
imp[hash[x].i][hash[x].j] > lastconstr(x)) {
              imp[hash[x].i][hash[x].j] =
firstconstr(x);
              x++;
              if (x > nn) end = TRUE;
              else imp[hash[x].i][hash[x].j]++;
            }
            if (x > nn+1 || (x==nn+1 &&
imp[hash[x].i][hash[x].j]==lastconstr(x))) end = TRUE;
else x=0;
            if (!end) checkanddo(x);
          }
        }
      }
    }
    printf("Total, %d solutions.\n",nsol);
    printf("The best score: %d\n",bestscore);
  printf("The end of searching.\n");
}
```

IV. Conclusion

The contrapositive is the property that makes the task hard. Even so, most implications found are weak. If other properties are not checking and one uses weaker equivalence tables, there are more interesting implications. Therefore, from a number of values on, it is a matter of choice. The well-known 3-valued logics have just a few of those checked properties.

In this version, the C/C++ programmer is the same person who uses the program. For further work, one can write a lexical analyzer and a parser in the same program, which will make it much larger than the exemplified in this paper. Thus, instead of placing comments in the source code, recompile and run the program, the properties can be written by the user in the input data file, whereas the program interpret them while checking. In this way, the constraint programming will be at a higher level. Certainly, the search will waste much longer time as the search is an exponential problem wrt time, but that will be a choice. Another possibility for future work consists in permitting search over conjunction and/or equivalence tables, which will certainly make the computation exponentially longer. For other possible existing methods, if any, a comparative analysis can be presented in the same work. The choice of 6 values is for making the explanation simpler, the previous testing feasible and the paper be shorter.

Acknowledgment

Many thanks to Des Watson, Rudi Lutz, Samson Abramsky and Stephen Eglen for their presence during the author's PhD research.

References

[1] U. Ferreira, *Uncertainty and a 7-Valued Logic*. ICCSA-2006, In Pradip Peter Dey, Mohammad N. Amin, and Thomas M. Gatton (Editors), Proceedings of The 2nd International Conference on Computer Science and its Applications, National University, San Diego, CA, USA, June, pp. 170-173. Now also at https://www.ulissesferreira.uk/A4-ValuedLogicCRC.pdf.

[2] J. Łukasiewicz, *On Three-Valued logic*. (1920) In L. Borkowski (Editor), Selected Works by Jan Łukasiewicz,, (1970) North-Holland, Amsterdam, pp. 87-88.

[3] S. C. Kleene, *Introduction to Metamathematics*. 1952, North-Holland, Publishing Company, Amsterdam.

[4] N. Belnap, *A Useful Four-Valued Logic*. In J. Michael Dunn and G. Epstein (Editors), Proceedings of the Fifth International Symposium on Multiple-Valued Logic. Modern Uses of Multiple-Valued Logic. Indiana University, D. Reidel Publishing Company, pp. 8-37.

[5] J. U. Ferreira, *A Four-Valued Logic*. CSITEC-2017, In Natarajan Meghanathan and David C. Wylds (Editors), 3rd International Conference on Computer Science, Information Technology, 9th International Conference on Networks & Communications (NeCoM-2017), pp. 71-84, Dubai, United Arab Emirate. Now also at https://www.ulissesferreira.uk/unand7v.pdf.