WUSS Paper 067-2022

True is not False: Evaluating Logical Expressions

Ronald J. Fehd, senior maverick, theoretical programmer,
Fragile-Free Software Institute

Abstract

description: The SAS® software language provides methods to evaluate logical expressions which then allow conditional execution of parts of programs. In cases where logical expressions contain combinations of intersection (and), negation (not), and union (or), later readers doing maintenance may question whether the expression is correct.

purpose: The purpose of this paper is to provide a truth table of Boole's rules, De Morgan's laws, and sql joins for anyone writing complex conditional statements in data steps creating subsets, merges with in=, macros, or procedures with a where clause. In my own work, taking the time to construct a truth table has helped me explain to my customers why their description of a subset is incomplete because of not reviewing the excluded observations.

audience: programmers, intermediate to advanced users

keywords: Boolean algegra, Boolean logic, De Morgan's laws, evaluation, logical operators, sql joins

In this paper

Introduction

overview

This paper combines the ideas of three logicians, Boole, De Morgan and Venn with the language of set theory and sql in order to assemble a table of logical expressions which describe each of the four permutations of pairs of true and false values.

The intent of this exercise is to provide a thesaurus for programmers who have specifications written by non-programmers.

The introduction contains these topics.

- natural language
- set theory
- sql
- comparison operators
- combinations, permutations
- four sets
- overlapping operators

---

natural language

Each natural language has a set of grammar rules about conjunctions that are used to describe pairs of ideas.
This is a list of common phrases; logical operators are in `text font`.

phrase	operator	logic	join	union
both ... and	`and`	disjunction	inner	intersection
either ... or but not both	`xor`, exclusive		left, right	except
either ... or	`or`, inclusive	conjunction	full	union
neither ... nor	`nor`			

Note: The words *also* and *only* are used in oral and written descriptions.

---

set theory

Set theory has four descriptions: union, intersection, set difference and symmetric difference.

phrase	Boolean	written	spoken
intersection	`and`	$A \cap B$	A cap B
set difference	`xor(T,F)`	$A \setminus B$	A and not B A minus B
symmetric difference	`xor`	$A \Delta B$	A xor B
union	`or`	$A \cup B$	A cup B

---

sql

Structured Query Language (sql) has two groups of operators, joins and unions.

Note: Some dialects of sql insert the word *outer* between the keywords *left, right, full* and *join*; e.g.: `full outer join` is equivalent to `full join`.

type	operator	Boolean	description
joins	`inner`	`and`	only in both tables
	`left`	`xor(T,F)`	only in left
	`right`	`xor(F,T)`	only in right
	`full`	`or`	all from both tables
unions	`except`	`xor(T,F)`	compare to left join
	`intersect`	`and`	compare to inner join
	`union`	`or`	compare to full join

---

comparison operators

A logical expression is evaluated according to the rules of Boolean algebra. Expressions with comparisons, or relations, are reduced to the Boolean set of values `(0,1)` with this set of operators.

parentheses:	evaluate expression inside			
equality:	equal:	`eq, =`		
	not equal:	`ne, ^=`		
quantity:	less than:	`lt, <`	less than or equal:	`le, <=`
	greater than:	`gt, >`	greater than or equal:	`ge, >=`

Note: Other programming languages refer to this concept as *relational operators*.

---

combinations, permutations

The difference between a combination and a permutation is a very important idea in deconstructing logical expressions. The *permutations* of two values `T` and `F` are two sets: `(T,F)` and `(F,T)`; but these two sets are different examples of the single *combination* `(T,F)`.

---

four sets

This table list the four permutations of two expressions,
Left L, and Right R, each with two values, true `T`, and false `F`.

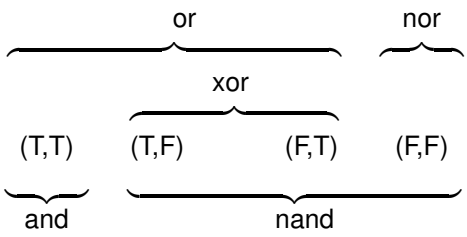
L	R
T	T
T	F
F	T
F	F

The task is to use Boole's operators and De Morgan's laws to uniquely identify each permutation.

---

overlapping operators

This set diagram shows which permutations return true for each operator. Notice that the operators `and` and `nor` define only one element as *True*, whereas `or`, `xor` and `nand` have two or more elements defined as *True*.



This list provides the operators with common natural language constructs and explanations.

or	inclusive or:	one, or more, **True**	
nor	not or:	neither **True**	*both* values **False**
xor	exclusive or:	only one **True**	but not both (and)
and	*both* **True**;	neither **False**	
nand	not and:	one, or more, **False**	

Venn diagrams of logical expressions

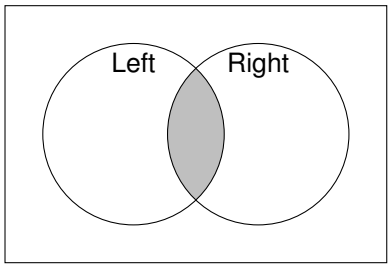
overview

John Venn was an English logician known for the visual representations of set theory known as Venn diagrams. The diagrams shown below illustrate the three operators `and`, `xor` and `or` with these three permutations of *true* and *false* values.

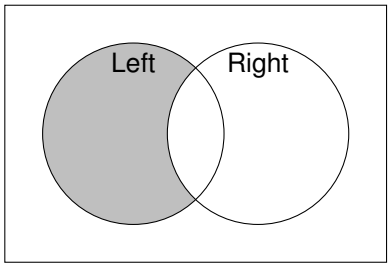
$$\underbrace{\underbrace{(T,T)}_{and}\quad\underbrace{(T,F)\quad(F,T)}_{xor}}_{or}$$

---

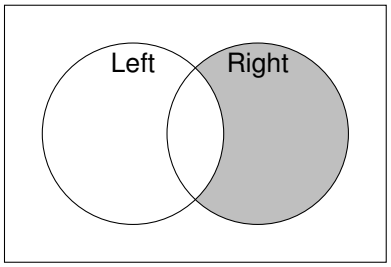
name : and
phrase : both ... and
expression : `L and R`
logic : disjunction
join : inner
union : intersect



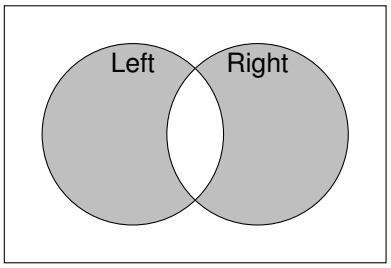
: xor-left
: only one
: but not the other
: `L and not R`
: left join



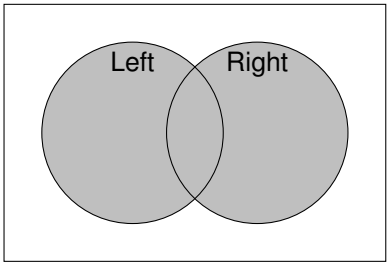
: xor-right
: only one
: but not the other
: `not L and R`
: right join



name : exclusive or
phrase : either ... or
: but not both
expression : `bxor(L,R)`
union : (L union R)
: except
: (L intersect R)



name : inclusive or
phrase : either ... or
expression : `L or R`
logic : conjunction
join : full



Truth tables of logical expressions

overview

This section contains the following topics.

- overlapping sets
- expressions
- De Morgan's laws

---

overlapping sets

This table shows the four permutations of sets of pairs of values, *and(T,T),* *xor-left(T,F),* *xor-right(F,T),* *nor(F,F),* and the logical operators *xor*, *or*, and *nand* which include two or more of the basic four.

name	values		
and	T,T		
xor-left	T,F	xor	or
xor-right	F,T	xor	or
nor	F,F		nor

nand

---

expressions

This table shows the logical expressions that are used to describe each of the four permutations of pairs of `(T,F)` values.

____ nand ____			values				
___ and ___	____ or ____	____ and ____	L	R	__ xor __	_ or _	_ nor, or _
		L and R	T	T		L or R	
not(L and R)	not L or not R	L and not R	T	F	bxor(L,R)	L or R	
not(L and R)	not L or not R	not L and R	F	T	bxor(L,R)	L or R	
not(L and R)	not L or not R	not L and not R	F	F			not(L or R)

nor, and

De Morgan's Laws

Augustus De Morgan was a contemporary of Boole. These laws are stated in formal logic.

Conjunction means *and*; *disjunction* means *or*.

nand : The negation of a conjunction is the disjunction of the negations.
nor : The negation of a disjunction is the conjunction of the negations.

name	expression	parentheses
nand: not and	`not L or not R`	no
	`not (L and R)`	required
nor: not or	`not L and not R`	no
	`not (L or R)`	required

Programs

truth table

This program shows a truth table of the logical expressions defined above and their resolution.

```
Title3 "Truth Table with L and R";
%let sysparm = 1,0;*boolean;

PROC format; value TF 0 = 'F'
                      1 = 'T';

DATA truth_table;
     label  nand_and        = 'nand-and; not(L and R)'
            nand_or         = 'nand-or; not L or not R'
            L               = 'L'
            R               = 'R'
            and_L_R         = 'and(T,T)'
            and_L_not_R     = 'xor-left; and(T,F)'
            and_not_L_R     = 'xor-right; and(F,T)'
            and_not_L_not_R = 'nor-and; and(not(F),not(F))'
            xor             = 'xor(T,F); xor(F,T)'
            or              = 'or(L,R)'
            nor_and         = 'nor-and; not L and not R'
            nor_or          = 'nor-or; not(L or R)';
     format _numeric_ TF.;
do    L = &sysparm;
   do R = &sysparm;
      nand_and        = not(L and R);
      nand_or         = not L or  not R;
      and_L_R         =     L and     R;
      and_L_not_R     =     L and not R;
      and_not_L_R     = not L and     R;
      and_not_L_not_R = not L and not R; *** duplicate;
      xor             = bxor(L,R);
      or              = L or R;
      nor_and         = not L and not R; *** duplicate;
      nor_or          = not(L or R);
      output;
      end;
   end;
stop;
run;
PROC print data  = &syslast label noobs
           split = ';';
```

output

```
nand.and      nand.or                          xor-left  xor-right  nor.and   xor(T,F)           nor.and   nor.or
not(L and R)  not L or not R   L  R   and(T,T)  and(T,F)  and(F,T)   and(F,F)  xor(F,T)  or(L,R)  and(F,F)  not(L or R)
-----------   --------------   -  -   -------  --------  ---------  --------  --------  -------  --------  ----------
          F                F   T  T   T        F         F          F         F         T        F         F
          T                T   T  F   F        T         F          F         T         T        F         F
          T                T   F  T   F        F         T          F         T         T        F         F
          T                T   F  F   F        F         F          T         F         F        T         T
                                                                    duplicate                     duplicate
```

true is not false

overview

This section contains programs for the following topics.

- data step function ifc
- implicit evaluation in macro expressions
- refactoring macro values with %eval

---

data step function ifc

The ifc function has four parameters:

1. a logical expression
2. character value returned when true
3. value returned when false
4. value returned when missing, which is optional

This example shows the function in a data step.

```
*name: ifc-data-values.sas;
%let false = false <---<<<;
data test_ifc_values;
     attrib value    length = 4
            text_if  length = $ %length( false)
            text_ifc length = $ %length(&false);
do value = ., -1 to 2;
   if not value then text_if = 'false';
   else              text_if = 'true';
   text_ifc = ifc(value, 'true'
                         , "&false"
                         , 'missing' );
   output;
   end;
stop;
run;
proc print data = &syslast noobs;
run;
```

```
value      text_if    text_ifc
-------    -------    -------------
    .       false     missing
   -1       true      true
    0       false     false <---<<<
    1       true      true
    2       true      true
```

Note zero and missing are false,
negative and positive values are true!

---

implicit evaluation in macro expressions

This program shows that the macro language performs an evaluation of an integer, similar to the data step function ifc.

```
%macro test_tf;
%do value = -1 %to 2;
    %if &value %then
        %put &=value is true;
    %else
        %put &=value is false;
    %end;
%mend;
%test_tf;
```

```
VALUE=-1 is true
VALUE= 0 is false
VALUE= 1 is true
VALUE= 2 is true
```

refactoring macro values with %eval

Many programmers provide a macro variable to use while debugging or testing. This macro variable may be initialized to any number of values representing *false*, such as *(no, off,)*, etc. The problem of checking for the correctly spelled value such as *YES, Yes, yes, Y, y, ON, On, on,* can be eliminated by recoding the value to boolean with this expression,

! → `%eval(0 ne &testing)` *or* `%eval(not(0 eq &testing))` which acknowledges any value other than zero as *true*.

```
 /*name: demo-macro-test-true-false.sas
%macro testing
      (testing=0 /* default: false, off */
      );
%*recode: any value turns testing on;
%let testing = %eval(0 ne &testing);
%if &testing %then %put &=testing is true;
%else              %put &=testing is false;
%mend;
```

```
10  %testing()
              TESTING=0 is false
11  %testing(testing=1)
              TESTING=1 is true
12  %testing(testing=.)
              TESTING=1 is true
13  %testing(testing=?)
              TESTING=1 is true
14  %testing(testing=T)
              TESTING=1 is true
15  %testing(testing=True)
              TESTING=1 is true
16  %testing(testing=yes)
              TESTING=1 is true
17  %testing(testing=no)
              TESTING=1 is true
```

Summary

Suggested Reading

people : [4], George Boole [3], Augustus De Morgan [5], John Venn

predecessors : Fehd [8] provides examples of checking command-line options during testing to add additional code to programs;
Fehd [9], using %sysfunc with ifc;
Fehd [10], macro design ideas

wiki : [6], set theory; [7], history of concept of truth table

sql : Fogleman [11], using sql except operator for a report similar to compare procedure;
Schreier [16], review of sql set operators outer union, union, intersect, and except with Venn diagrams;
ren Lassen [15], Lassen to SAS-L about sql xor;
Lafler [14], Lafler, sql, beyond basics, 3e;
Celko [1], J. Celko on sql relational division;
[2], sql null values

Venn diagrams in SAS : Harris [12], macros to create 2-, 3-, and 4-way Venn diagrams
Kruger [13], macro using Google

Conclusion

Evaluating logical expressions has two aspects, conversion of comparisons to boolean values and logical algebra using the operators `not`, `and` and `or`. This paper provides the following benefits. The names of each of the permutations are *and, xor-left, xor-right,* and *nor*. Each permutation is identified using `not` with `and`. The names of sets of permutations are *xor, or,* and *nand*. Venn diagrams are provided for each of the permutations; these visual representations are helpful in understanding the sql concepts of *addition* and *subtraction* of the permutations. With this vocabulary and conceptual representations a programmer may be confident of understanding requirements and specifications no matter what language, discipline or scientific dialect they are written in.

Author Information

Ronald J. Fehd Ron.Fehd.macro.maven at gmail.com

About the author:
LinkedIn `https://www.linkedin.com/in/ronald-fehd-5125991/`
affiliation senior maverick, theoretical programmer, Fragile-Free Software Institute
also known as `macro maven` on SAS-L

Acknowledgements

Kirk Lafler, Søren Lassen, Zoriana Kurzeja and Nat Wooding reviewed a draft of this paper; each provided clarification on sql concepts. Lassen noted his SAS-L post with reference to Celko's sql explanation. Kurzeja noted that sql nulls are not the same as SAS missing values; see [2].

Trademarks

References

[1] Joe Celko. Divided we stand: The sql of relational division. In *Simple Talk*, 2009. URL https://www.simple-talk.com/sql/t-sql-programming/divided-we-stand-the-sql-of-relational-division/.

[2] W3school Staff et al. Sql null values, 2019. URL https://www.w3schools.com/sql/sql_null_values.asp.

[3] Wikipedia Editors et al. Augustus De Morgan. In *Wikipedia, The Free Encyclopedia*, 2016. URL https://en.wikipedia.org/wiki/Augustus_De_Morgan.

[4] Wikipedia Editors et al. George Boole. In *Wikipedia, The Free Encyclopedia*, 2016. URL https://en.wikipedia.org/wiki/George_Boole.

[5] Wikipedia Editors et al. John Venn. In *Wikipedia, The Free Encyclopedia*, 2016. URL https://en.wikipedia.org/wiki/John_Venn.

[6] Wikipedia Editors et al. Set theory. In *Wikipedia, The Free Encyclopedia*, 2016. URL https://en.wikipedia.org/wiki/Set_theory.

[7] Wikipedia Editors et al. Truth table. In *Wikipedia, The Free Encyclopedia*, 2019. URL https://en.wikipedia.org/wiki/Truth_table.

[8] Ronald J. Fehd. Writing testing-aware programs that self-report when testing options are true. In *North-East SAS Users Group Conference Proceedings*, 2007. URL http://www.lexjansen.com/nesug/nesug07/cc/cc12.pdf. Coders' Corner, 20 pp.; topics: options used while testing: echoauto, mprint, source2, verbose; variable testing in data step or macros; call execute; references.

[9] Ronald J. Fehd. Using functions Sysfunc and Ifc to conditionally execute statements in open code. In *SAS Global Forum Annual Conference Proceedings*, 2009. URL http://support.sas.com/resources/papers/proceedings09/054-2009.pdf. Coders Corner, 10 pp.; topics: combining functions ifc, nrstr, sysfunc; assertions for testing: existence of catalog, data, file, or fileref; references.

[10] Ronald J. Fehd. Macro design ideas: Theory, template, practice. In *SAS Global Forum Annual Conference Proceedings*, 2014. URL http://support.sas.com/resources/papers/proceedings14/1899-2014.pdf. 21 pp.; logic, quality assurance, testing, style guide, documentation.

[11] Stanley Fogleman. Teaching a new dog old tricks — using the except operator in proc sql and generation data sets to produce a comparison report. In *MidWest SAS Users Group Conference Proceedings*, 2006. URL www.lexjansen.com/nesug/nesug06/cc/cc10.pdf. Beyond Basics, 3 pp.; using sql except to produce report similar to compare procedure.

[12] Kriss Harris. V is for Venn diagrams. In *PharmaSUG China Conference Proceedings*, 2018. URL https://www.lexjansen.com/pharmasug-cn/2018/DV/Pharmasug-China-2018-DV29.pdf. 21 pp.; macros using Graph Template Language (gtl) to procude 2-, 3-, and 4-way Venn diagrams.

[13] Hillary Kruger. Creating proportional Venn diagrams using google. In *SAS Global Forum Annual Conference Proceedings*, 2011.

[14] Kirk Paul Lafler. *PROC SQL: Beyond the Basics Using SAS(R), Third Edition*. SAS Institute, 2019. URL https://www.sas.com/store/books/categories/usage-and-reference/proc-sql-beyond-the-basics-using-sas-third-edition/prodBK_71650_en.html.

[15] Søren Lassen. Re: Excellent short tutorial on sql. In *SAS-L archives*, 2016. URL https://listserv.uga.edu/cgi-bin/wa?A2=SAS-L;aa4234fb.1604b.

[16] Howard Schreier. SQL set operators: So handy Venn you need them. In *SAS Users Group International Annual Conference Proceedings*, 2006. URL www2.sas.com/proceedings/sugi31/242-31.pdf. Tutorials, 18 pp.; outer union, union, intersect, and except.