

Paper BB-186

True is not False: Evaluating Logical Expressions

Ronald J. Fehd, Stakana Analytics

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, macros, or procedures with a where clause. Audience: programmers, intermediate to advanced users Keywords: Boolean algebra, Boolean logic, De Morgan's law, evaluation, logical operators, sql joins
-----------------	---

In this paper	Introduction	1
	Venn Diagrams of Logical Expressions	4
	Truth Tables of Logical Expressions	5
	Programs	6
	Truth Table	6
	True is not False	7
	Summary	8
	References	9

Introduction**Overview**

This article 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

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	xor, exclusive		left, right	except
but not both				
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, ^= (caret=), ~= (tilde=)

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

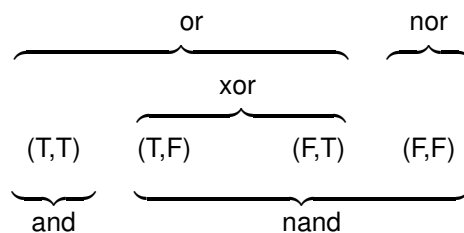
This table lists the four permutations of two expressions, Left L, and Right R, each with two values, true T, and false F.

<u>L</u>	<u>R</u>
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**

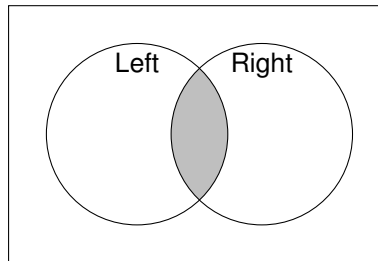
Venn Diagrams of Logical Expressions

Overview

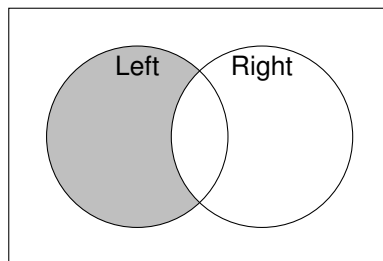
John Venn was an English logician known for the visual representations of set theory known as Venn diagrams. The diagrams shown below illustrate the three operators `and`, `xor` and `or` with these three permutations of *true* and *false* values.

$$\underbrace{\begin{matrix} (T, T) \\ \text{and} \end{matrix}}_{\text{or}} \quad \underbrace{\begin{matrix} (T, F) & (F, T) \\ \text{xor} \end{matrix}}$$

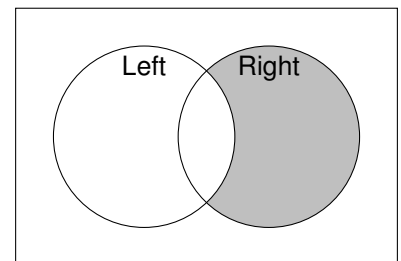
name : and
 phrase : both ... and
 expression : L and R
 logic : disjunction
 join : inner
 union : intersect



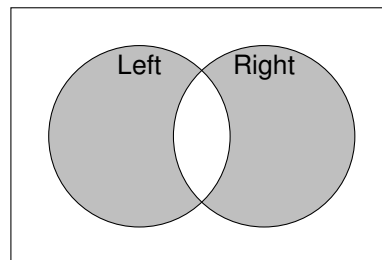
: xor-left
 : only one
 : but not the other
 : L and not R
 : left join



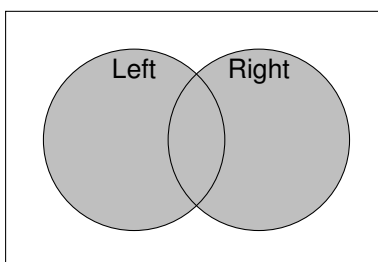
: xor-right
 : only one
 : but not the other
 : not L and R
 : right join



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	
nand	{	xor-left	T,F
		xor-right	F,T
		nor	F,F
	}		

expressions

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

_____ nand _____		_____ and _____		_____ and _____		values		_____ xor _____	_____ or _____	_____ nor, or _____
_____ and _____	_____ or _____	_____ and _____	_____ or _____	L and R	L and not R	L	R			
				T	F	T	F		L or R	
not(L and R)	not L or not R			L and not R	not L and R	T	F	bxor(L,R)	L or R	
not(L and R)	not L or not R			not L and R	not L and not 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 rules 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.

```

16 Title3 "Truth Table with L and R";
17 %let sysparm = 1,0;*boolean;
18
19 PROC format; value TF 0 = 'F'
20                    1 = 'T';
21
22 DATA truth_table;
23     label  nand_and      = 'nand-and; not(L and R)'
24           nand_or       = 'nand-or; not L or not R'
25           L             = 'L'
26           R             = 'R'
27           and_L_R       = 'and(T,T)'
28           and_L_not_R   = 'xor-left; and(T,F)'
29           and_not_L_R   = 'xor-right; and(F,T)'
30           and_not_L_not_R = 'nor-and; and(not(F),not(F))'
31           xor           = 'xor(T,F); xor(F,T)'
32           or            = 'or(L,R)'
33           nor_and       = 'nor-and; not L and not R'
34           nor_or        = 'nor-or; not(L or R)';
35     format _numeric_ TF.;
36     do L = &sysparm;
37         do R = &sysparm;
38             nand_and      = not(L and R);
39             nand_or       = not L or not R;
40             and_L_R       = L and R;
41             and_L_not_R   = L and not R;
42             and_not_L_R   = not L and R;
43             and_not_L_not_R = not L and not R; *** duplicate;
44             xor           = bxor(L,R);
45             or            = L or R;
46             nor_and       = not L and not R; *** duplicate;
47             nor_or        = not(L or R);
48             output;
49         end;
50     end;
51 stop;
52 run;
53 PROC print data = &syslast label noobs
54             split = ' ';

```

output

	nand.and not(L and R)	nand.or not L or not R	L	R	and(T,T)	xor-left and(T,F)	xor-right and(F,T)	nor.and and(F,F)	xor(T,F) xor(F,T)	or(L,R)	nor.and and(F,F)	nor.or not(L or R)
1												
2												
3												
4		F	F	T	T	F	F	F	F	T	F	F
5		T	T	T	F	T	F	F	T	T	F	F
6		T	T	F	T	F	T	F	T	T	F	F
7		T	T	F	F	F	F	T	F	F	T	T
8								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.

```

1  %let false = false <---<<<;
2  DATA test_ifc_integers;
3      attrib integer length = 4
4          text_if length = $ %length( false)
5          text_ifc length = $ %length(&false);
6  do integer = ., -1 to 2;
7      if not integer then text_if = 'false';
8      else text_if = 'true';
9      text_ifc = ifc(integer, 'true'
10                  , "&false"
11                  , 'missing' );
12  output;
13  end;
14  stop;
15  run;
16
17  PROC print data = &syslast noobs;
```

integer	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.

```

1  %macro test_tf;
2  %do value = -1 %to 2;
3      %if &value %then
4          %put &=value is true;
5      %else
6          %put &=value is false;
7      %end;
8  %mend;
9  %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 comparison expression,

! → %eval(0 ne &testing)
which acknowledges any value other than zero as *true*.

<pre> 1 %macro testing 2 (testing=0 /* default: false, off */ 3); 4 %*recode: any value turns testing on; 5 %let testing = %eval(0 ne &testing); 6 %if &testing %then %do; 7 %put &=testing is true; 8 %end; 9 %else %do; 10 %put &=testing is false; 11 %end; 12 %mend;</pre>	<pre> 1 10 %testing() 2 TESTING=0 is false 3 11 %testing(testing=1) 4 TESTING=1 is true 5 12 %testing(testing=.) 6 TESTING=1 is true 7 13 %testing(testing=?) 8 TESTING=1 is true 9 14 %testing(testing=T) 10 TESTING=1 is true 11 15 %testing(testing=True) 12 TESTING=1 is true 13 16 %testing(testing=yes) 14 TESTING=1 is true 15 17 %testing(testing=no) 16 TESTING=1 is true</pre>
--	--

Summary

Suggested Reading	people : [3], George Boole [2], Augustus De Morgan [4], John Venn predecessors : [6] shows three logical and expressions to choose output data sets [7] provides examples of checking command-line options during testing to add additional code to programs [8], using %sysfunc with ifc [9], macro design ideas sets : [5] set theory sql : [10], using sql except operator for a report similar to compare procedure; [13], review of sql set operators outer union, union, intersect, and except with Venn diagrams; [12]Lassen to SAS-L about sql xor; [11] Lafler, sql, beyond basics, 2e; [1]J. Celko on sql relational division
---------------------------------	--

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.

Contact Information

Ronald J. Fehd

Ron.Fehd.macro.maven@gmail.com

About the author:

sco.wiki

http://www.sascommunity.org/wiki/Ronald_J._Fehd

LinkedIn

www.linkedin.com/Ronald.Fehd

affiliation

Stakana Analytics, Senior Maverick

also known as

macro maven on SAS-L, Theoretical Programmer

Programs:

<http://www.sascommunity.org/wiki/>

Evaluating Logical Expressions

Truth Table

Writing Testing Aware Programs

Acknowledgements

Kirk Lafler and Søren Lassen 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.

Trademarks

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. In the USA and other countries ® indicates USA registration. Other brand and product names are trademarks of their respective companies.

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] Wikipedia Editors et al. Augustus De Morgan. In *Wikipedia, The Free Encyclopedia*, 2016. URL https://en.wikipedia.org/wiki/Augustus_De_Morgan.
- [3] Wikipedia Editors et al. George Boole. In *Wikipedia, The Free Encyclopedia*, 2016. URL https://en.wikipedia.org/wiki/George_Boole.
- [4] Wikipedia Editors et al. John Venn. In *Wikipedia, The Free Encyclopedia*, 2016. URL https://en.wikipedia.org/wiki/John_Venn.
- [5] Wikipedia Editors et al. Set theory. In *Wikipedia, The Free Encyclopedia*, 2016. URL https://en.wikipedia.org/wiki/Set_theory.
- [6] Editor R.J. Fehd. Macro Extract. In *sasCommunity.org*, 2008. URL http://www.sascommunity.org/wiki/Macro_Extract. given two snapshots, extract database adds, changes, deletes.
- [7] Ronald J. Fehd. Writing testing-aware programs that self-report when testing options are true. In *NorthEast SAS Users Group Conference Proceedings*, 2007. URL <http://www.nesug.org/Proceedings/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.
- [8] 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.
- [9] 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.; topics: logic, quality assurance, testing, style guide, documentation, bibliography.
- [10] 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.
- [11] Kirk Paul Lafler. *PROC SQL: Beyond the Basics Using SAS(R), Second Edition*. SAS Institute, 2013. URL http://www.sas.com/store/prodBK_62432_en.html.
- [12] 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>.
- [13] 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.