Pace University
DigitalCommons@Pace

CSIS Technical Reports

Ivan G. Seidenberg School of Computer Science and Information Systems

8-1-2004

A Review of Logic for Computer Professionals

Ronald I. Frank
Pace University

Follow this and additional works at: http://digitalcommons.pace.edu/csis_tech_reports

Recommended Citation
Frank, Ronald I., "A Review of Logic for Computer Professionals" (2004). *CSIS Technical Reports.* Paper 16.
http://digitalcommons.pace.edu/csis_tech_reports/16

This Article is brought to you for free and open access by the Ivan G. Seidenberg School of Computer Science and Information Systems at DigitalCommons@Pace. It has been accepted for inclusion in CSIS Technical Reports by an authorized administrator of DigitalCommons@Pace. For more information, please contact rracelis@pace.edu.

SCHOOL OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

TECHNICAL REPORT

Number 204, August 2004

A Review of Logic for Computer Professionals

Ronald I. Frank

PACE UNIVERSITY

Ronald I. Frank

Information Technology Researcher and Innovator, Senior R & D Project Manager, International Consultant, R & D Business Chairman & CEO, and Life-long Educator.

IBM

I. Research Division staff member, manager & corporate-board consultant
II. Principal IBM US spokesman/consultant for the engineering and scientific PC
III. Scientific Center staff member / project manager and staff member & Computer Center Manger
IV. District and Branch Offices Scientific Marketing Rep.
V. Service Bureau Development Programmer: commercial & scientific applications

IBM Projects

I. Corporate-board consultant [educational uses of the internet]
II. Research manager of parallel computer interfaces
III. Research manager of low-end APL
IV. Research staff member (APL language and Virtual Reality research
V. Team created the first Remote Operation of [370/3090 Main Frame] Computers
- VM OS Programmable Operator" component.

VI. Team created first modern LAN Network Management.
VII. Team developed the first WAN line quality monitoring technology.
VIII. APL Development team - Earliest "Home Use" Of Personal Computers
- Developed Scientific and Engineering applications for "proto personal computers".

IX. Manager IBM Systems Research Institute Computing Center [Grad School & Research Center]
X. Scientific Marketing Consultant to ~10% of US Industrial Research. (Esso (sic), ATT, Bell Labs, General Dynamics Electric Boat, etc.)
- Special Scientific rep to the University of Pennsylvania / Consultant to Provost on developing a computing facility for Academic/Scientific use
- Created some of the earliest courses and taught IBM-ers and customers what computers were and how they could be used.

Entrepreneurial Business Experience

IMMERSIVE SYSTEMS INC. (A virtual reality R & D company) Chairman and CEO

EDUCATION

Pace University School of Computer Science & Information Systems DPS in Computing.

New York University Courant Institute of Mathematical Sciences Ph.D. Student In Applied Math as an *IBM GRADUATE RESIDENT FELLOW. MS Applied Math & Ph.D.-"ABD".*

Rutgers University BA, **Math**

ACADEMIC EXPERIENCE (Undergraduate and Graduate Level)-

Massachusetts College System & Pace University All courses in CS & IS & Telecom.

HOBBIES

I like to hike the Westchester North County Trailway, snorkel, and read science fiction - not all at the same time.

A Review Of Logic For Computer Professionals

© Ronald I. Frank 2000, 2004

ABSTRACT

Logic is used in programming and circuit design. I introduce the basic definition of Logic and give some concrete examples.

By "Logic" I mean the representation and manipulation of variables that can be in one of two states. In programming the variables stand for conditional statements used in looping, that can be true or false. In circuit design the variables stand for voltage on or off or current flowing or not flowing.

A section exhibits some applications of Logic to computer language "if" and "while" statements.

The appendix shows an example of XOR applied to RAID storage, the proof of the sufficiency of Sheffer Stroke (NAND), and some examples from formal circuit theory. They are in the appendix because they require mathematical maturity (and some motivation) to fully follow them.

A bibliography points to standard texts for more detailed information.

This paper is not meant to substitute for the bibliographic references. It is meant as a quick and broad introduction, and as a short pertinent reference for classes using these topics.

TABLE OF CONTENTS

A REVIEW OF LOGIC FOR COMPUTER PROFESSIONALS 1

ABSTRACT 1

TABLE OF CONTENTS 2

I. Preliminary Ideas And Terminology 3
A). Two-state Variables (TSV) 3
B). Functions Of Two-state Variables (TSVF) 3
C). Truth Table Representation Of Functions Of Two-state Variables 3
Figure 1: Truth Table Examples Of Basic Two Two-state Variable Functions 5
Figure 2: The 16 Logical Functions of Two Boolean Variables 6

II. Logical Functions As Circuit 7
A). [AND, OR, NOT] 7
Figure 3: Series Parallel Circuits [AND, OR, NOT] 7
Figure 4: XOR as a circuit 7
B). XOR As A Circuit 7
C). DeMorgan's Laws As Circuits 8
Figure 5: DeMorgan's Laws As Circuits 8
D). NAND As A Circuit 9
Figure 6: NAND As A Circuit 9

III. Application Of Logic To Programming Language Conditionals 10
A). [if] 10
Figure 7: Tracing The Simple if-else Conditional Statement 10
B). [while] 11
Figure 8: Tracing The Simple while Conditional Statement 11

IV. Appendices 12
A). Some Laws Of XOR [⊕] And Its Application To RAID Storage 12
1) DEFINITION OF "Exclusive OR", Some Of Its Properties 12
2) Practice Manipulating ⊕: 12
3) Application to RAID Storage 13
B). Proof Of The Sufficiency Of Sheffer Stroke 14
C). Formal Circuit Elements 14
Figure 9: Circuit Element Symbols; NOT, OR, & AND 15
Figure 10: Half Adder Cases & Some Full Adder Cases 16
Figure 11: Full Adder Practice 17

IV Bibliography 18

I. Preliminary Ideas And Terminology

A). Two-state Variables (TSV)

It's always easier to think in terms of canonical examples. So, instead of being abstract and talking of two-state variables, we will think of variables that stand for statements that are true or false. Later, we can think of 0 voltage as false and high voltage as true (or vice versa) and apply our new knowledge to circuits.

If I state that P is a statement that is true or false, I am being half abstract since I am not giving an actual statement that is true or false. We can think of a canonical example: "I worked more than 40 hours this week." This is a statement. It is either true or false.

B). Functions Of Two-state Variables (TSVF)

There are a bunch of useful functions of <u>two</u> two-state variables that we regularly use in computing. If P is a TSV and Q is too (ha ha), then we can represent their possible separate values as P=0 or P=1 and Q=0 or Q=1.

Variables that can take on only two values are called Boolean variables. In fact this whole topic of formalization of classical logic was started by George Boole in the century before last.

If P stands for "I worked more than 40 hours this week" and Q stands for (represents) "I got paid for only 40 hours work", then the two variable function

P AND Q

represents the TSVF "I worked more than 40 hours this week and I got paid for only 40 hours work." This compound statement is also a TSVF since it is either true or false.

WEIIII .... Maybe.

We have to agree exactly on what is meant by the language conjunction "and".

There has been an interaction between language and Logic so that today we have very precise meanings for words like AND, OR, and IMPLIES, all of which are two Two-state Variable Functions. The meanings are the ones defined in Logic.

C). Truth Table Representation Of Functions Of Two-state Variables

A Truth Table is a standard form display of the input variables (TSVs) and the output of the function (a TSV). The function is symbolized and put in a header – see below for examples of AND, OR, NOT, IMPLIES, TRUE, and FALSE.

AND, OR, IMPLIES, TRUE, and FALSE are two-variable functions. TRUE is the TVF that gives the value 1 (true), no matter what the input variable values are.

Similarly, FALSE is the TVF that gives the value 0 (false), no matter what the input variable values are.

NOT, however, is a one-variable function.

The symbols for the functions are fairly common ones coming from the Latin word for the functions. $\wedge$ means AND, and $\vee$ symbolizes (means) OR.

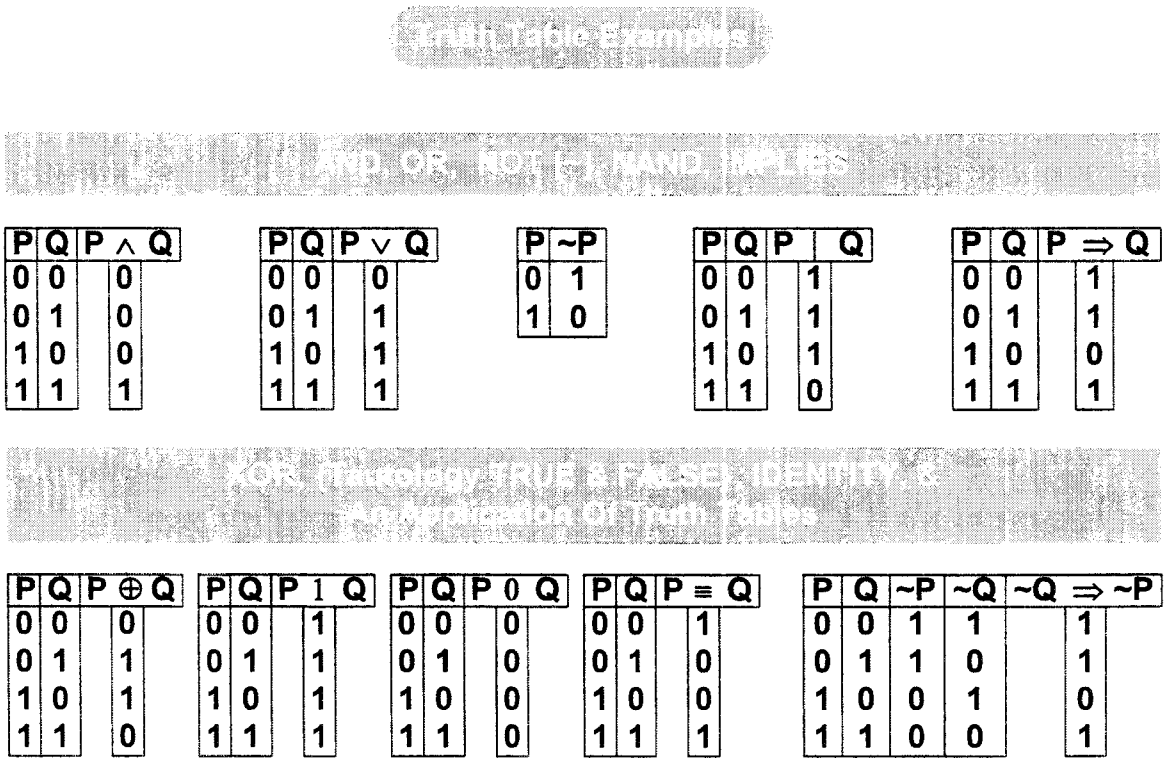
NAND (symbolized by the Sheffer "Stroke", or stile character) is used in circuits. There is also a NOR function.

IMPLIES is meant to capture some of the meaning of implication in language. "All humans are mortal AND I am a human IMPLIES I am mortal". In logic, we take IMPLIES to mean something a little stronger the usual use of implies in language. If you look at the truth table for $\Rightarrow$, which is the symbol for implies, you see that IMPLIES is false ONLY when the premise is TRUE and the conclusion is FALSE. The implication (ho ho) of this is that, from the table, we see FALSE IMPLIES anything. Assume a falsehood, and I you can validly imply anything I want. The general format of a Truth Table is:

Input 1	Input 2	Output Function
Value 1	Value 1	Output 1
Value 2	Value 2	Output 2
Value 3	Value 3	Output 3
Value 4	Value 4	Output 4

Where all input values are TRUE or FALSE and each output is TRUE or FALSE

Figure 1: Truth Table Examples Of Basic Two Two-state Variable Functions



Notice NAND is the NOT of AND. NOR is the NOT of OR, [work it out or see Figure 2 below]. The last application example shows how to add helper columns to ease table construction.

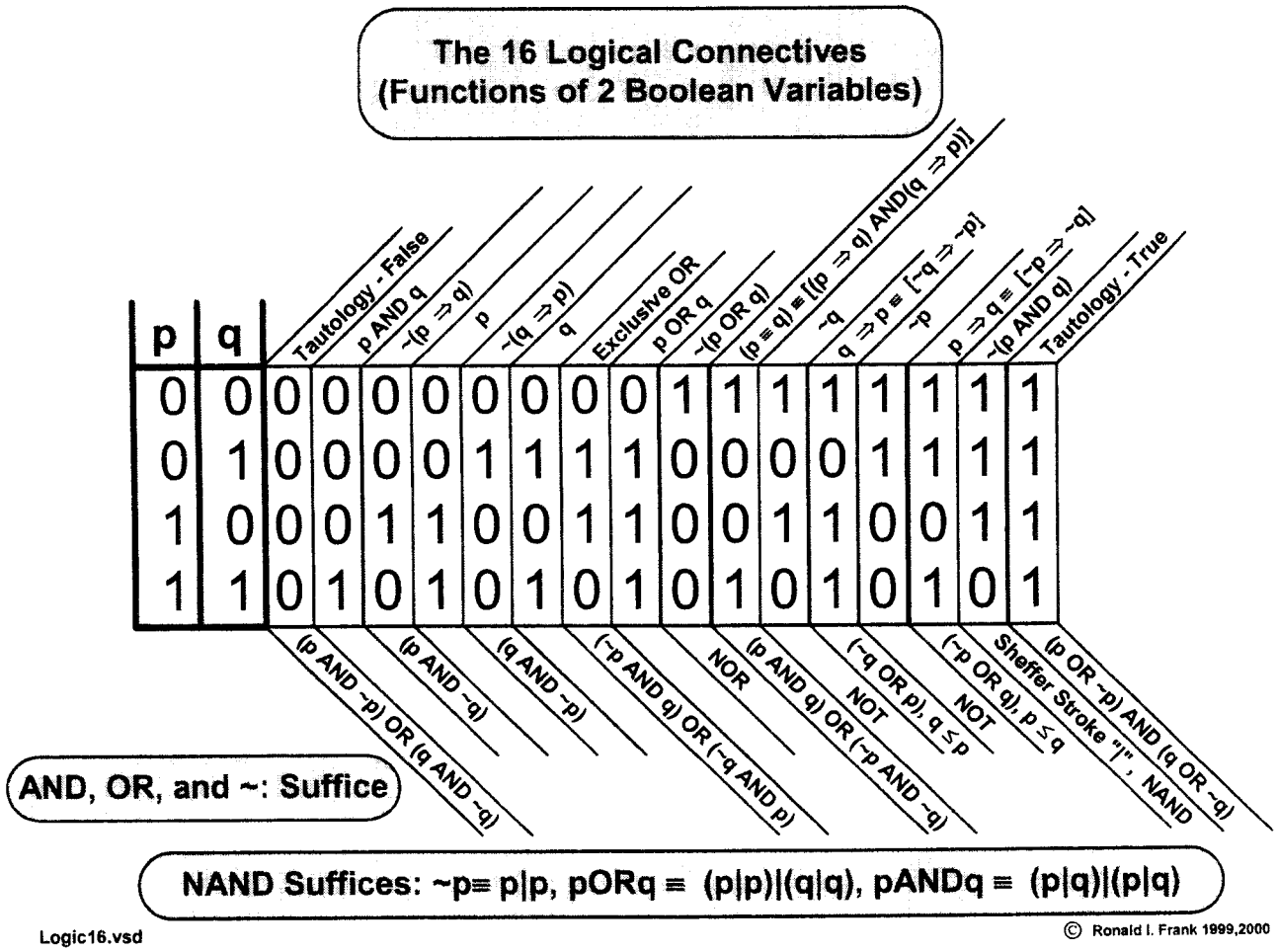
Notice that the output of a two variable function is itself a TSV in that only the values 1 and 0 appear. Since there are 4 input cases [0 0, 0 1, 1 0, 1 1] there are 4 possible output cases of 0 or 1. This means that there are 16 possible two variable functions [2x2x2x2=16 possible outcomes].

I present for your reference, the 16 cases and their symbolic interpretation.

NOTE: Every one of the 16 possible two variable functions can be represented using only the functions AND, OR , and the NOT. We say that AND, OR, and NOT suffice to generate all the others.

Similarly the Sheffer stroke alone(!) can generate all of the others! The implication of this is that a single circuit element can build all logic circuits! [The same can be said of the logic function NOR]/

Figure 2: The 16 Logical Functions of Two Boolean Variables



Notice the groupings of the columns [# of Columns = total of 16]:

1. [1] NOR is NOT of OR.
2. [1] NAND is the NOT of AND.

3. [2] p output reproduces the TRUE inputs of p. NOT p reproduces the FALSEs.
4. [2] q output reproduces the TRUE inputs of q. NOT p reproduces the FALSEs.

5. [1] Output TRUE is TRUE for all possible inputs.
6. [1] Output FALSE is FALSE for all possible inputs.

7. [2] AND and OR (and the NOTs) have no further breakdown into equivalents.

8. [1] IMPLICATION ($p \Rightarrow q$) has equivalents and can be defined as $p \leq q$.
9. [1] CONTRAPOSITIVE ($q \Rightarrow p$) has equivalents and can be defined as $q \leq p$.
10. [2] The NOT of IMPLICATION AND CONTRAPOSITIVE have equivalents.

11. [1] IDENTITY ($\equiv$) is when the inputs are equal.

12. [1] EXCLUSIVE OR ($\oplus$) is often used in terms of its equivalent expressions.

II. Logical Functions As Circuit

A). [AND, OR, NOT]

If we think of a simple lever switch that can make or break a contact, we can represent some of the logical functions as simple circuits with switches in them. Some of the functions require double throw switches – that can make or break on either a left throw or right throw of the switch.

Figure 3: Series Parallel Circuits [AND, OR, NOT]

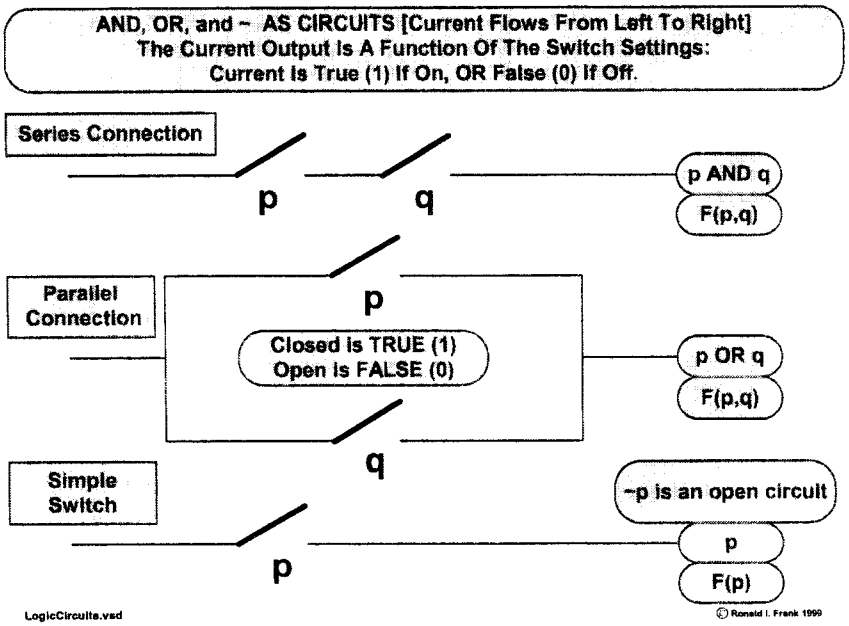
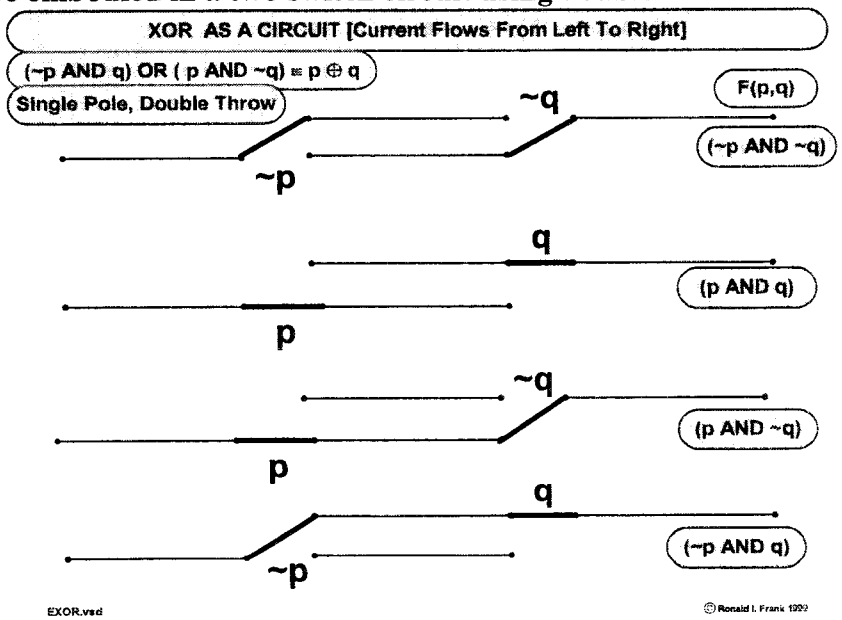


Figure 4: XOR as a circuit

B). XOR As A Circuit

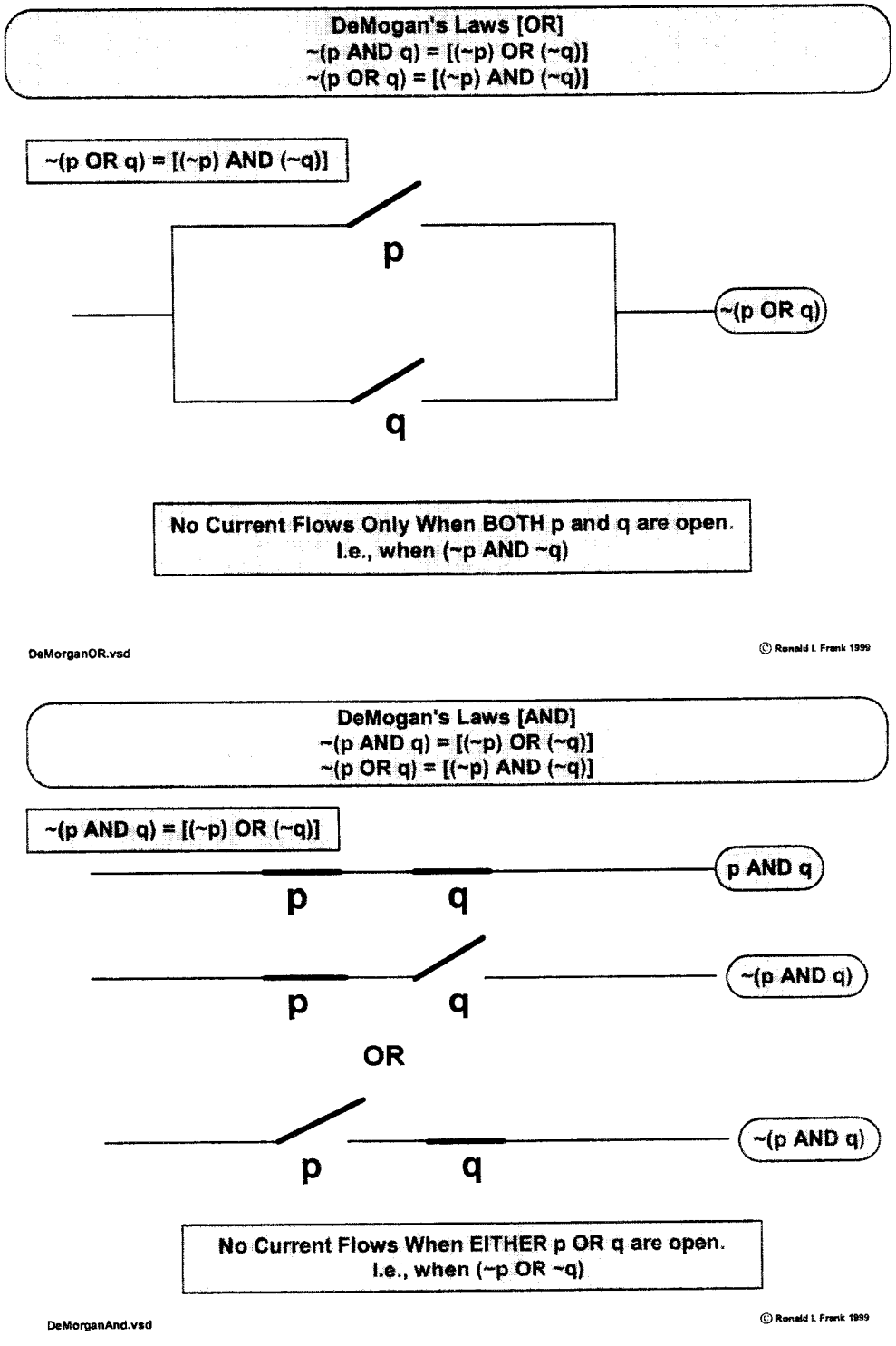
Exclusive or can be embodied in a two switch circuit using double throw switches.



C). DeMorgan's Laws As Circuits

There are a pair of very useful relationships, or laws, used to convert the negations of ANDs to ORs and vice versa.

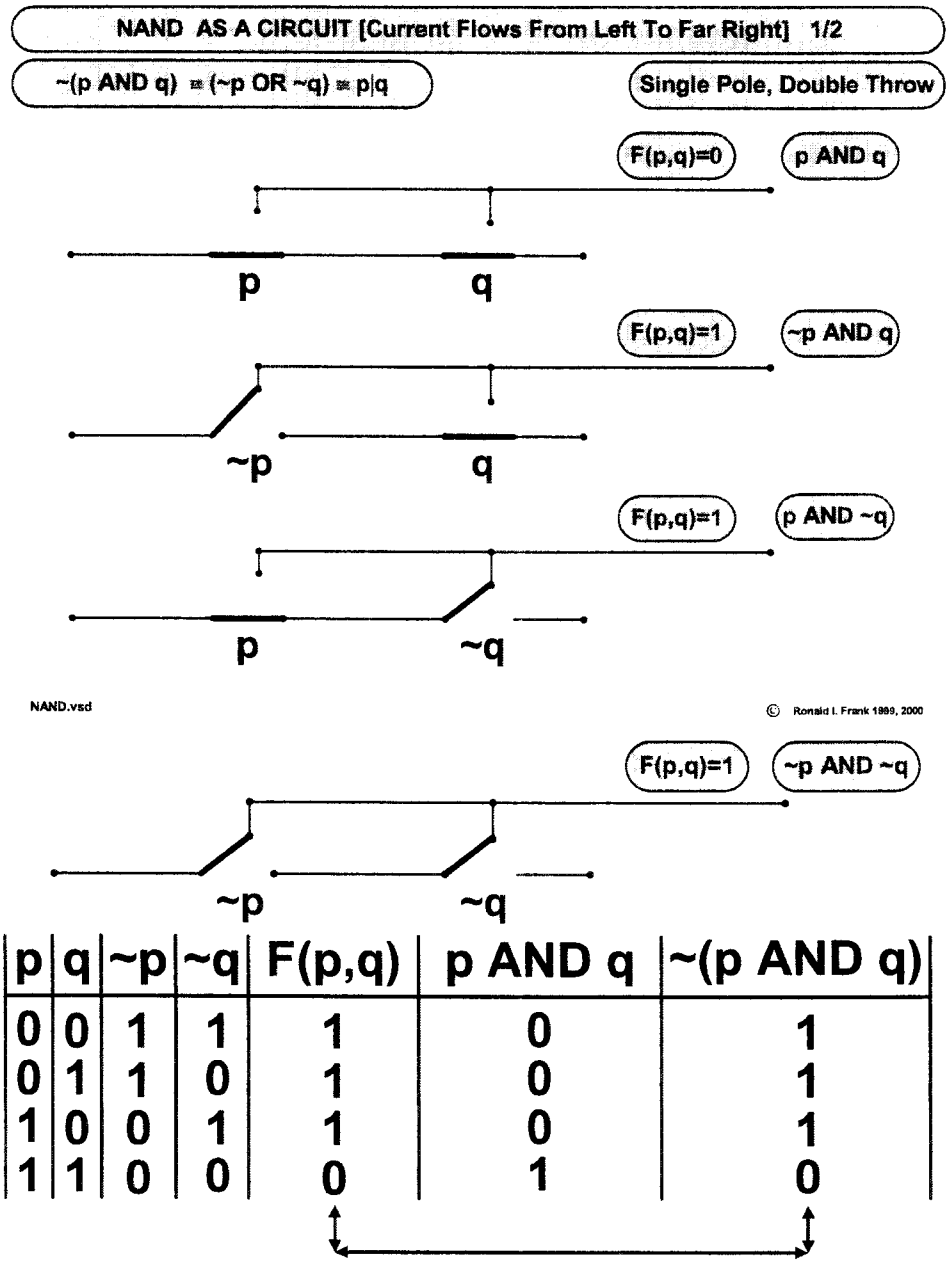
Figure 5: DeMorgan's Laws As Circuits



D). NAND As A Circuit

NOT AND can also be embodied in a two switch circuit using double throw switches.

Figure 6: NAND As A Circuit



p	q	~p	~q	F(p,q)	p AND q	~(p AND q)
0	0	1	1	1	0	1
0	1	1	0	1	0	1
1	0	0	1	1	0	1
1	1	0	0	0	1	0

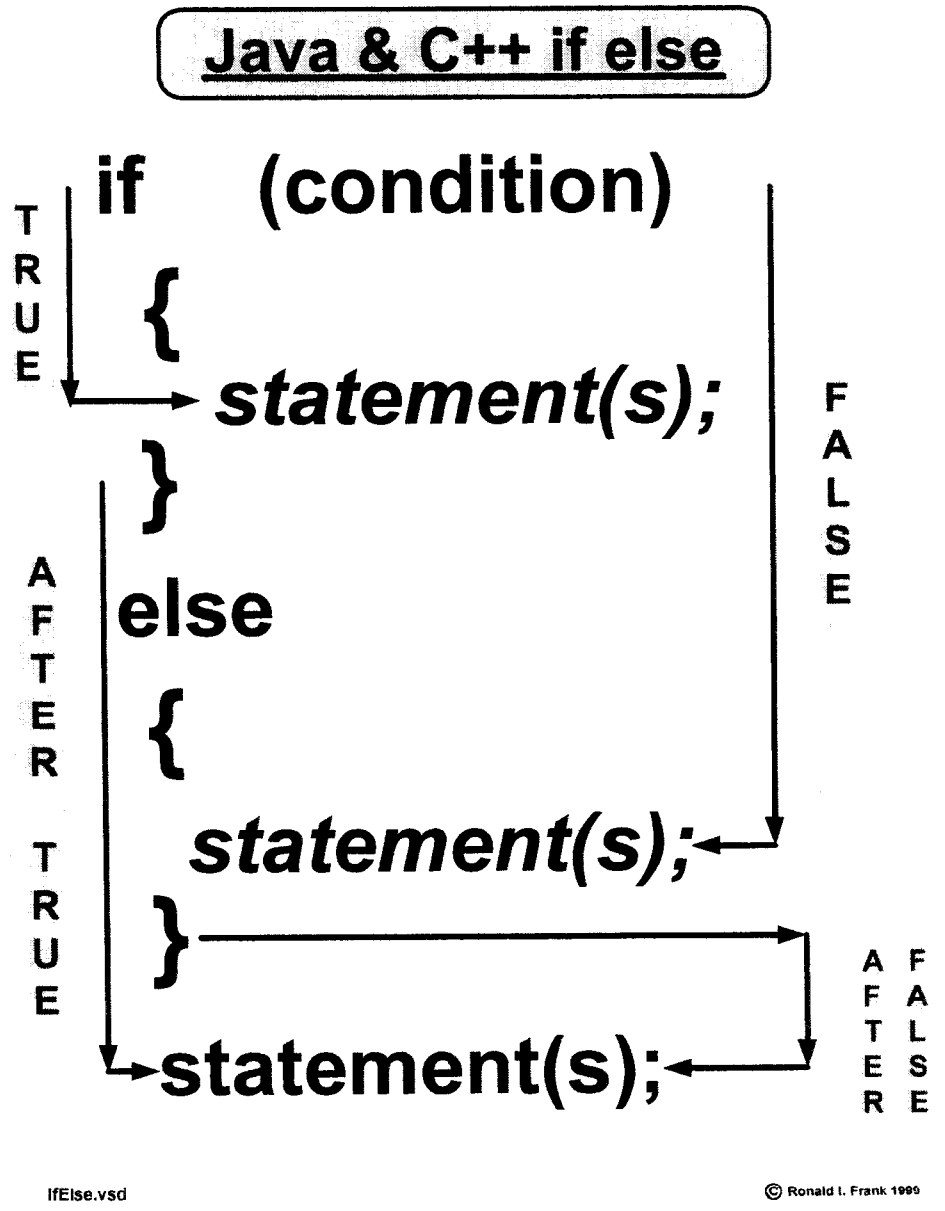
III. Application Of Logic To Programming Language Conditionals

A). [if]

The Java and C, C++ if-statement depends on a conditional (a statement that is TRUE or FALSE).

The languages allows the use of AND (&&), OR (||), and NOT (!) in the statement of the conditional. As we have seen above, this means any of the logical functions can be programmed into the conditional, including exclusive OR.

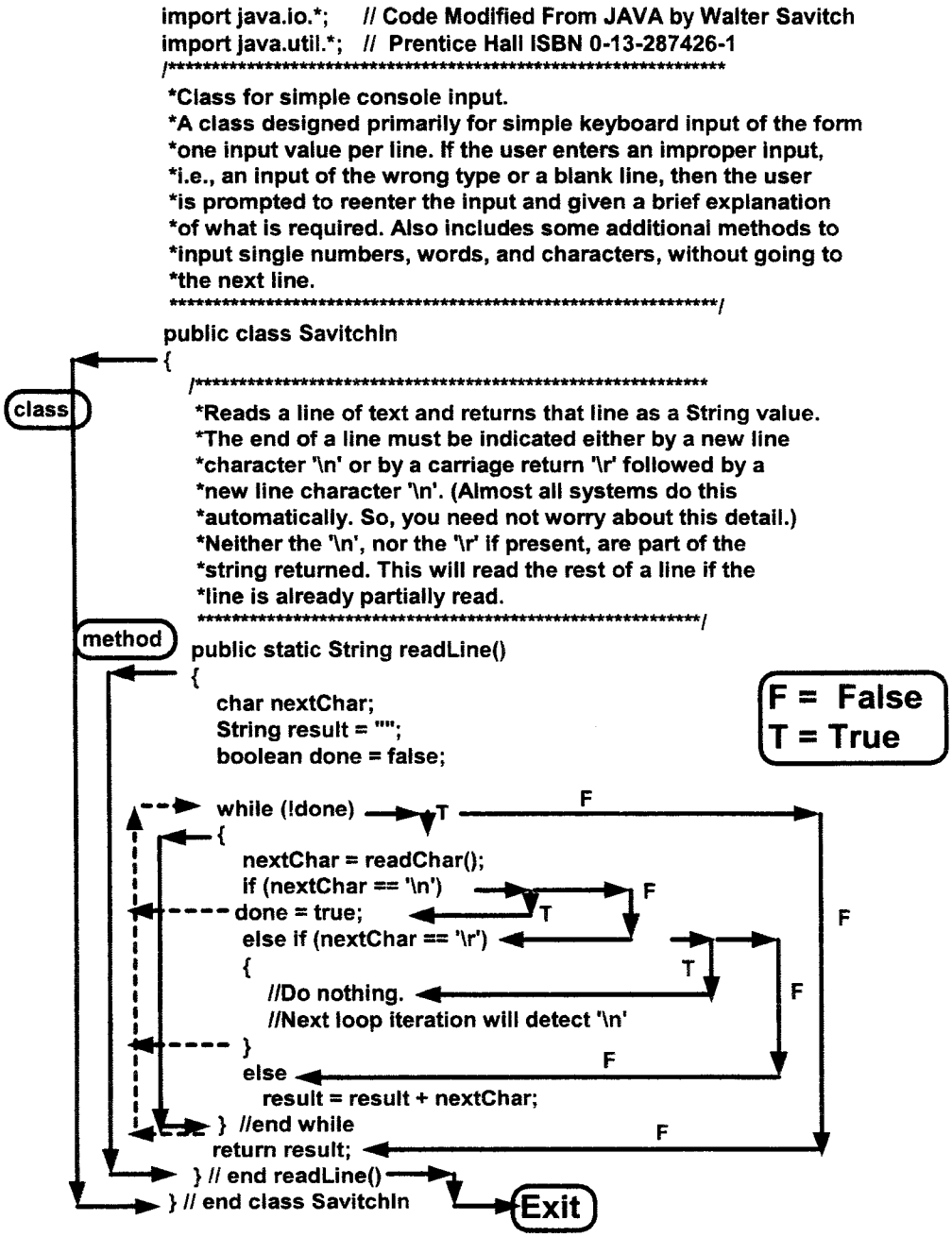
Figure 7: Tracing The Simple if-else Conditional Statement



B). [while]

The Java and C, C++ while-statement depends on a conditional (a statement that is TRUE or FALSE). The languages allows the use of AND (&&), OR (||), and NOT (!) in the statement of the conditional. As we have seen above, this means any of the logical functions can be programmed into the conditional, including exclusive OR. This is taken out of a text book as indicated in its comments. I notated the flow.

Figure 8: Tracing The Simple while Conditional Statement.



SavReadInLogic.vsd

© Ronald I. Frank 1999, 2000

IV. Appendices

A). Some Laws Of XOR [⊕] And Its Application To RAID Storage

1) DEFINITION OF "Exclusive OR", Some Of Its Properties

Truth Table

P	*Q*	*P* ⊕ *Q*
0	0	0
0	1	1
1	0	1
1	1	0

Properties

[⊕ Can be thought of as binary addition without carry]
[A and B are equal length bit strings, 0 is all 0s, 1 is all 1s]

1) $A \oplus B = B \oplus A$ — Commutivity

2) $A \oplus B \oplus C = A \oplus (B \oplus C) = (A \oplus B) \oplus C$ — Associativity

3) *Not so* $A \oplus (B \wedge C) = (A \oplus B) \wedge (A \oplus C)$ — Distributivity Over ∧

4) *Not so* $A \oplus (B \vee C) = (A \oplus B) \vee (A \oplus C)$ — Distributivity Over ∨

5) $\sim(A \oplus B) = \sim A \oplus B = A \oplus \sim B$ — Negation

6) $A \oplus 0 = A$ $\quad A \oplus 1 = \sim A$ — "Identities"

7) $A \oplus A = 0$ — Self-"Inverse"

8) $A \oplus \sim A = 1$ — The Universe

9) $A \oplus B = (A \wedge \sim B) \vee (\sim A \wedge B)$ — Definition in terms of ∨, ∧, ~

---

2) Practice Manipulating ⊕:

Theorem: **For Any A and B, equal length bit strings**

$A \leftarrow A \oplus B$
$B \leftarrow A \oplus B$
$A \leftarrow A \oplus B$

Interchanges A and B

Proof:
$C \leftarrow A \oplus B$
$B \leftarrow D \leftarrow C \oplus B = (A \oplus B) \oplus B = A \oplus (B \oplus B) = A \oplus (0) = A$
$A \leftarrow C \oplus D = (A \oplus B) \oplus A = (A \oplus A) \oplus B = (0) \oplus B = B$

---

3) Application to RAID Storage

[Redundant Array Of Inexpensive (or Independent) Disks]

We are not told the manufacturers' algorithms for striping and computing parity. We are only told that they use parity. Here is a view of how that might be done. Assume one block [A] is on Disk 1, another block [B] is on Disk 2 and their byte-by-byte parity [P] is on Disk 3. This takes 1/2 the extra space of mirroring

Example:
A = [10010111]
B = [11100010]
P = [01110101] even parity.
$P = A \oplus B$

If A is lost
$P \oplus B = (A \oplus B) \oplus B = A \oplus (B \oplus B) = A \oplus (0) = A$

If B is lost
$P \oplus A = (A \oplus B) \oplus A = B \oplus (A \oplus A) = B \oplus (0) = B$

If P is lost
$P = A \oplus B$

Example:
A = [10010111]
B = [11100010]
Q = [10001010] odd parity.
Let P=~Q and proceed as above.

B). Proof Of The Sufficiency Of Sheffer Stroke

(To Generate All 16 Boolean Functions Of Two Boolean Variables)

Proof Of Sheffer Stroke (NAND) Sufficiency

NAND Suffices: ~p ≡ p|p, pANDq ≡ (p|q)|(p|q), pORq ≡ (p|p)|(q|q)

1) ~p ≡ ~(p AND p) ≡ p|p **By Definition**

2) (p AND q) ≡ ~ ~ (p AND q) ≡ ~(p|q) ≡ (p|q)|(p|q) **By 1) Above**

3) (p OR q) ≡ [~(~p)] OR [~(~q)] ≡ ~[(~p) AND (~q)] ≡ (~p)|(~q) ≡ (p|p)|(q|q)
By DeMorgan and 1) Above

C). Formal Circuit Elements

I present the symbols for a few logic (boolean) functions [AND, OR, XOR, and NOT].

Next is the diagram for all four cases of a half adder (one bit position sum plus carry out, but no carry in). This is NOT the only possible design for a half adder.

Below this is two of the eight cases of a full adder (one bit position sum plus carry out, and a carry in). There are eight cases because there are four possible values for the two addends [00, 01, 10, 11], and two possible values for the carry in [0 or 1] – giving eight cases.

There is a sheet of four blank full adder diagrams for the reader to practice on.
Copy the diagram sheet and do all eight cases. Good luck.

What is missing here (because I think it is too large a deviation from the simple purpose of this paper) is the derivation of the designs of the half and full adders from first principles (e.g., by using a Karnaugh map)

Figure 9: Circuit Element Symbols; NOT, OR, & AND

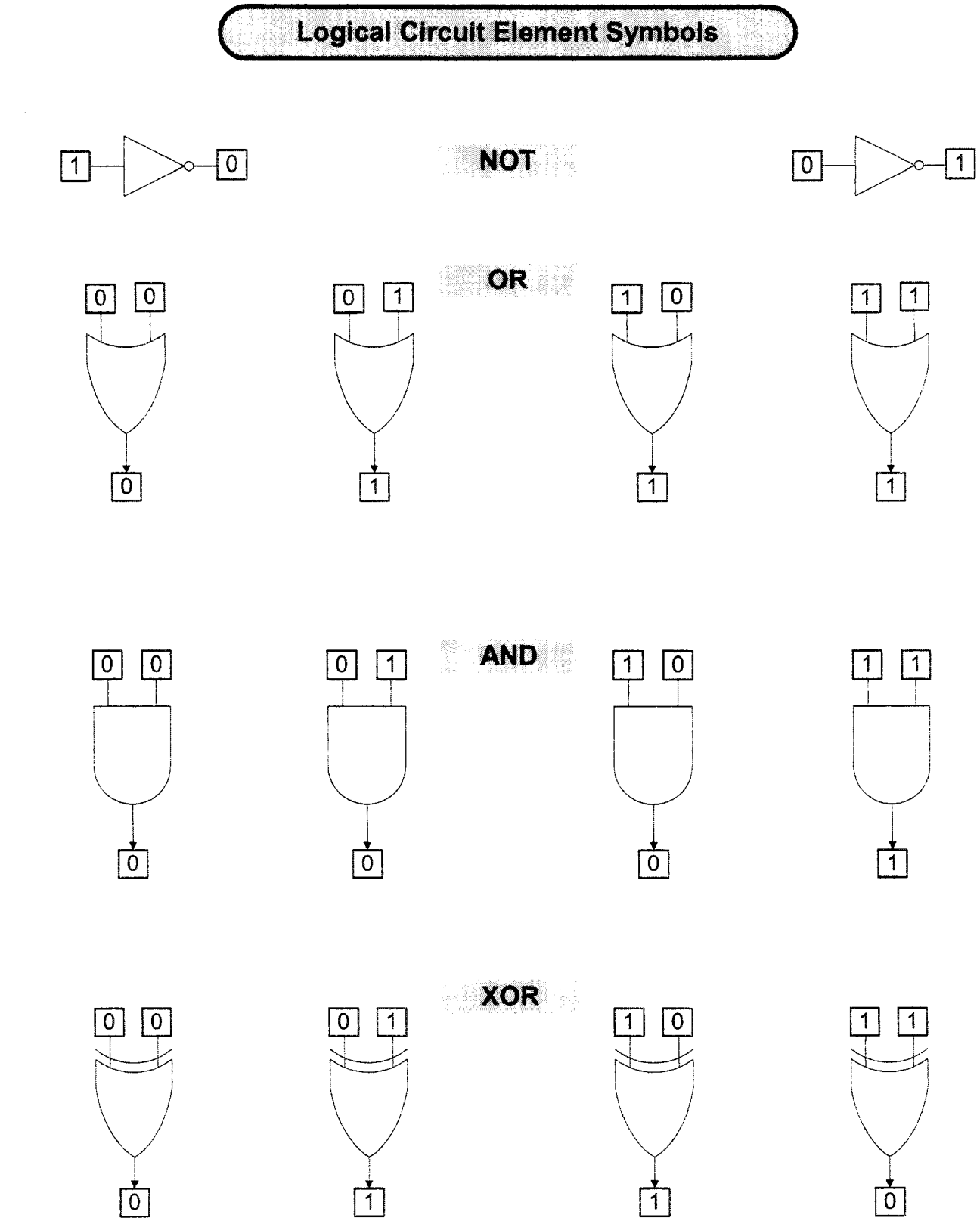


Figure 10: Half Adder Cases & Some Full Adder Cases

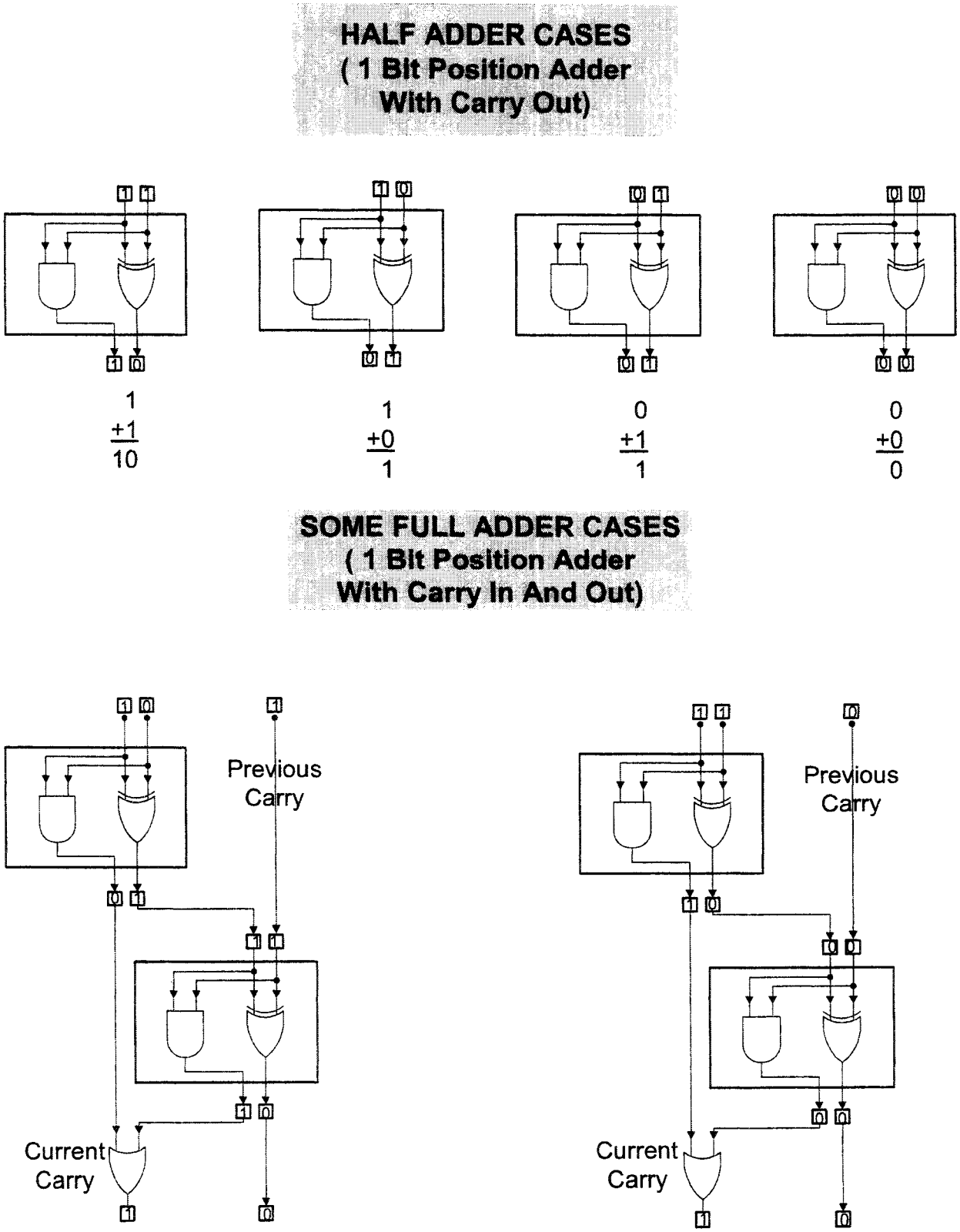
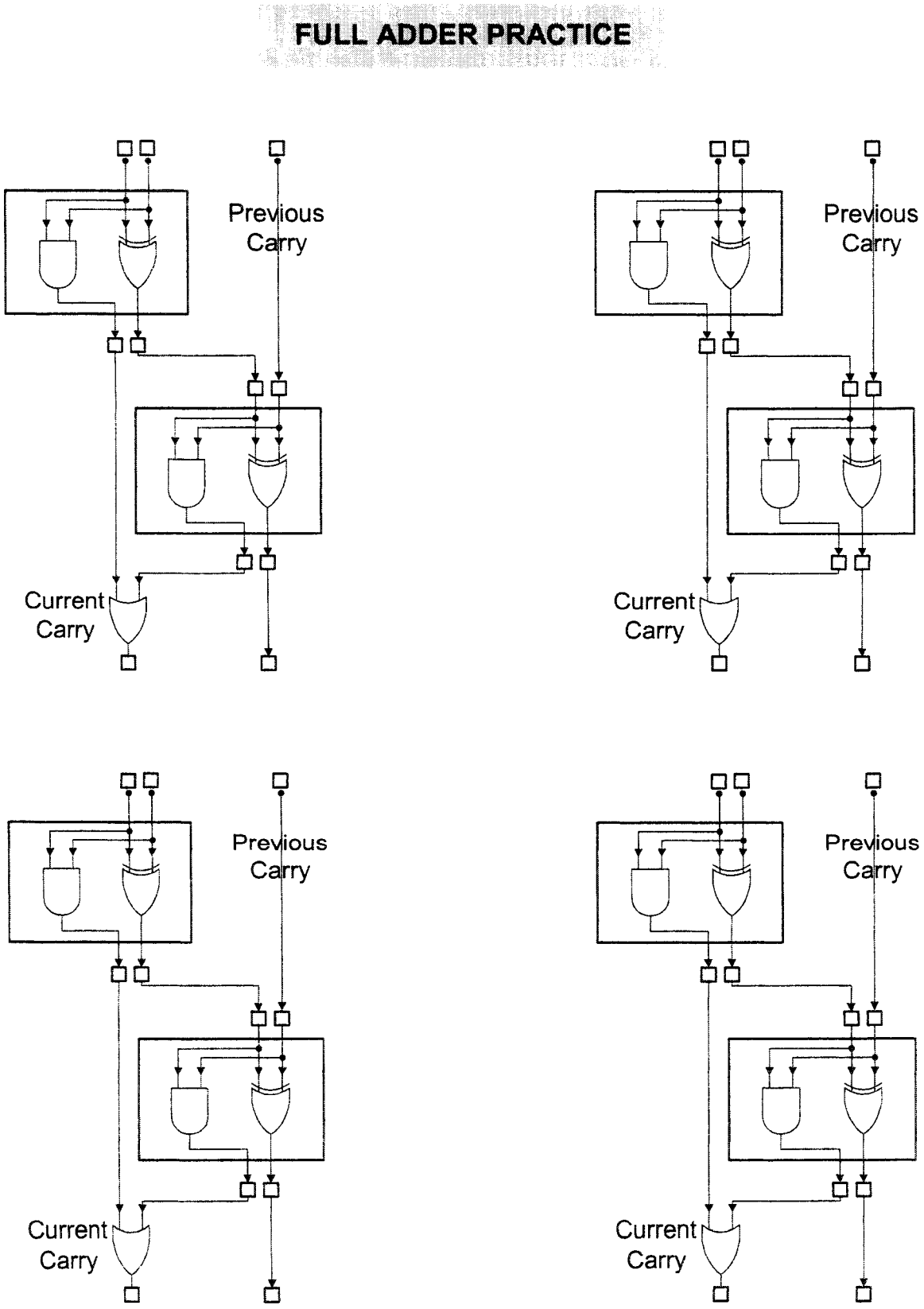


Figure 11: Full Adder Practice



IV Bibliography

[1] **Rosen, K. H., "Discrete Mathematics And Its Applications", McGraw-Hill (1995) ISBN 0-07-053965-0**

Especially pp. 608-650 on Boolean Algebra (and circuit applications)

[2] **Mano, M. M., "Computer System Architecture", 3rd Ed., Prentice Hall (1993) ISBN 0-13-175563-3**

Especially pp. 4-11 on Logic Gates and Boolean Algebra

o

School of Computer Science and Information Systems
Pace University
Technical Report Series

EDITORIAL BOARD

Editor:
Allen Stix, Computer Science, Pace--Westchester

Associate Editors:
Connie Knapp, Information Systems, Pace--New York
Susan M. Merritt, Dean, SCSIS--Pace

Members:
Howard S. Blum, Computer Science, Pace--New York
Donald M. Booker, Information Systems, Pace--New York
M. Judith Caouette, Office Information Systems, Pace--Westchester
Nicholas J. DeLillo, Mathematics and Computer Science, Manhattan College
Fred Grossman, Information Systems, Pace--New York
Fran Goertzel Gustavson, Information Systems, Pace--Westchester
Joseph F. Malerba, Computer Science, Pace--Westchester
John S. Mallozzi, Computer Information Sciences, Iona College
John C. Molluzzo, Information Systems, Pace--New York
Narayan S. Murthy, Computer Science, Pace--New York
Catherine Ricardo, Computer Information Sciences, Iona College
Sylvester Tuohy, Computer Science, Pace--Westchester
C. T. Zahn, Computer Science, Pace--Westchester

The School of Computer Science and Information Systems, through the Technical Report Series, provides members of the community an opportunity to disseminate the results of their research by publishing monographs, working papers, and tutorials. *Technical Reports* is a place where scholarly striving is respected.

All preprints and recent reprints are requested and accepted. New manuscripts are read by two members of the editorial board; the editor decides upon publication. Authors, please note that production is Xerographic from the pages you have submitted. Statements of policy and mission may be found in issues #29 (April 1990) and #34 (September 1990).

Please direct submissions as well as requests for single copies to:

Allen Stix
School of CS & IS - Suite 412 Graduate Center
Pace University
1 Martine Avenue
White Plains, NY 10606-1932