

Logic in Programming

Logic is extremely important in both the hardware and software of computing. Here we will begin with the software aspects of logic which are involved in programming. Later we will briefly show some hardware aspects of gates involved in computing architecture. Logic involves conditions in virtually all Choice and Loop constructs (If, and While forms). Logic also occurs in assertions, preconditions, post conditions, invariants and even comments.

The boolean (or logical) type is binary; it has only two values:

```
true false
```

Logical boxes (variables) include such examples as:

```
male, female, tall, old, done, isIsosceles
```

which often must be clarified, perhaps by comments.

For example, "old" could be defined as any age over 12 (or 21, 30, 100, whatever).

Declarations of logical types specifications of the two kinds of values; they have a form which begins with "**boolean**" followed by any number of variables separated by commas and end with a semicolon. Examples are:

```
boolean male, female;  
boolean old; // age over 12  
boolean tall; // height equal to or over 72 inches
```

Assignment of truth values to boolean boxes (variables) can be done simply as:

```
male = true;  
old = false;  
tall = old ;
```

Arithmetic Relations

Arithmetic relations often occur in logical conditions;
the relations compare two quantities of the same type (such as ints).

($a < b$) which reads "a is less than b"

($c > d$) which reads "c is more than d" or "c is greater than d"

($e \leq f$) which reads "e is greater than or equal to f"

($g \geq h$) which reads "g is greater than or equal to h"

($i == j$) which reads "i is equal to j"

($j != k$) which reads "j is not equal to k"

Alternate or equivalent ways are possible to express conditions:

($p < q$) is equivalent to ($q > p$)

for example

($\text{age} < 12$) is equivalent to ($12 > \text{age}$)

since they both are true for values 11, 10, 9, 8, .. etc

and both are false for values 12 and 13, 14, 15, ..., etc

Similarly, other equivalent conditions are:

($x \leq y$) is equivalent to ($y \geq x$)

as in the examples:

($\text{sum} \geq 7$) is equivalent to ($7 \leq \text{sum}$)

Notice especially that ($a < b$) is **not** the opposite or complement of ($a > b$).
Complements are treated shortly.

Assignment may also involve comparison with relations such as:

```
over21 = (age > 21);
tied   = (visitorScore == homeScore);
error  = (age < 0);
proper = (percent <= 100);
tall   = (height >= 72);    // inches
error2 = (denominator == 0);
```

Equivalent Control Forms: (optional at first)

Logical assignments can also be done as imperative or control-oriented forms.

For example the assignment

```
over21 = (age > 21);
```

can be written as the Choice form:

<pre>-- Jr PseudoCode If age > 21 Set over21 = true Else Set over21 = false EndIf</pre>	<pre>// Java Code if (age > 21) { over21 = true; } else { over21 = false; } //end if</pre>
--	---

Use of logical terms can be done in a number of ways. For example,

```
legal = (over21 == true);
```

can be done simpler by

```
legal = over21;
```

Complements (Opposites, negatives)

Complements are logical opposites; when one is true the complement is false. Examples of complements include male, female also tall, short and old, young.

Complements are expressed with the logical operator "!" which reads "not". The complement, or Not, is unary, it acts on the one condition that follows it. If the boolean box **b** is true then the complement **! b** is false; if **b** is false then **! b** is true, as show in the following truth table,

<u>bVal</u>	<u>! bVal</u>		<u>b</u>	<u>!b</u>
true	false	and a shorter version	T	F
false	true		F	T

For example, consider the assignment:

```
small = ! tall ;
```

If tall is true then small is false, and if tall is false then small is true.

Other example assignments follow:

```
female= ! male;
```

```
young = ! (age > 12);
```

Complements can involve arithmetic relations:

(a < b) is the complement of (a >= b)

(a > b) is the complement of (a <= b)

(a == b) is the complement of (a != b)

The above condition for young can be written without the not operator as:

```
!(age > 12) is (age <= 12)
```

Similarly:

```
!(age <= 21) is (age > 21)
```

Logical Binary Operations (And and Or)

Operations on logical or boolean boxes include two binary operators (And, and Or); The first letter of And and Or may be capitalized to avoid confusion, as in the line above.

And (also called "andAlso" or conjunction, symbolized by `&&`)

Or (also called "eitherOr" or disjunction, symbolized by `|` |)

Binary operations (Or, And) operate on two operands; the operator is between the two operands (called infix).

And is a logical operation often called the conjunction.

The And denoted `(p && q)` is true when p is true and q is true.

Examples of the use of this And follow.

```
increasing = (x < y) && (y < z);
```

```
equilateral = (s1 == s2) && (s1 == s3);
```

```
isInRange = (percent >= 0) && (percent <= 100);
```

```
isEligible = (over21) && (isEmployed);
```

Truth table of the logical And operator is shown below, including a shorter form; The And of two operands is true only in one case, when both p and q are true.

first	second	first && second	p	q	p && q
false	false	false	F	F	F
false	true	false	F	T	F
true	false	false	T	F	F
true	true	true	T	T	T

Or is a logical operation often called the disjunction. eitherOr, or inclusive-or; The Or denoted (p || q) is true when either p or q or both are true.

Examples of the use of the Or follow, in Java.

```
winpoint = (sum == 7) || (sum == 11);
error    = (percent < 0) || (percent > 100);
playBall = (inning <= 9) || (score1 == score2);
isosceles = (a == b) || (b==c) || (a==c);
```

Truth table of the logical Or operator is shown below.

The Or of two operands is true in 3 cases, when either p or q are true. The Or is false only in one case, when both p and q are false.

Or		
p	q	p q
F	F	F
F	T	T
T	F	T
T	T	T

Xor		
p	q	p ^ q
F	F	F
F	T	T
T	F	T
T	T	F

Exclusive-Or (or Xor) is similar to this common Or, but it is not often used. The exclusive-Or is true if only one or the other, but not both are true; The Exclusive-Or differs in the last row of the truth table as shown.

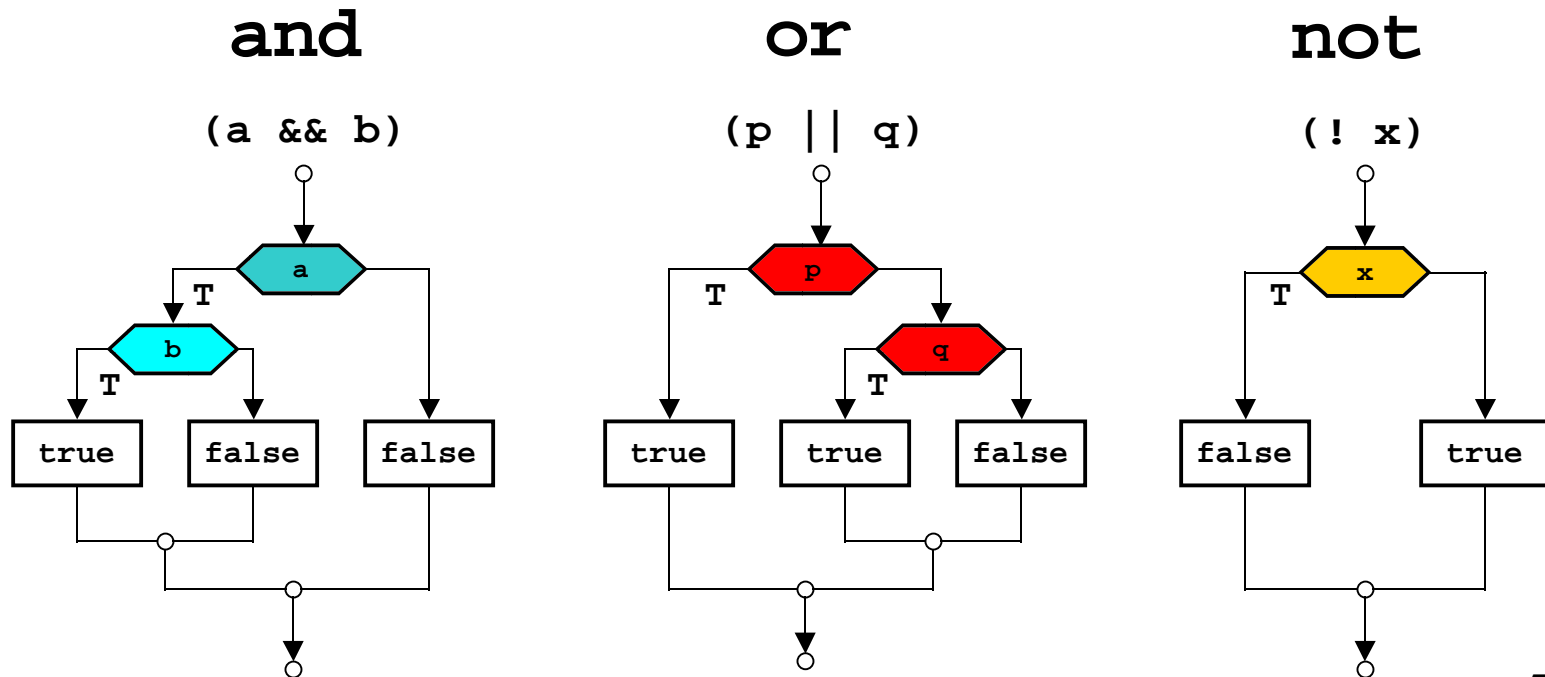
Logical Views:

Control Flow View (short circuit or Lazy way)

Behavior of logical operators can be shown by control flow charts as given below.

Note that if the first condition "a" of the And is false, then the second one "b" is not tested.
Note also that if the first condition of the Or is true then the second one is not tested.

In other words, sometimes both conditions are not tested;
this is called "short circuit" evaluation or "lazy evaluation".

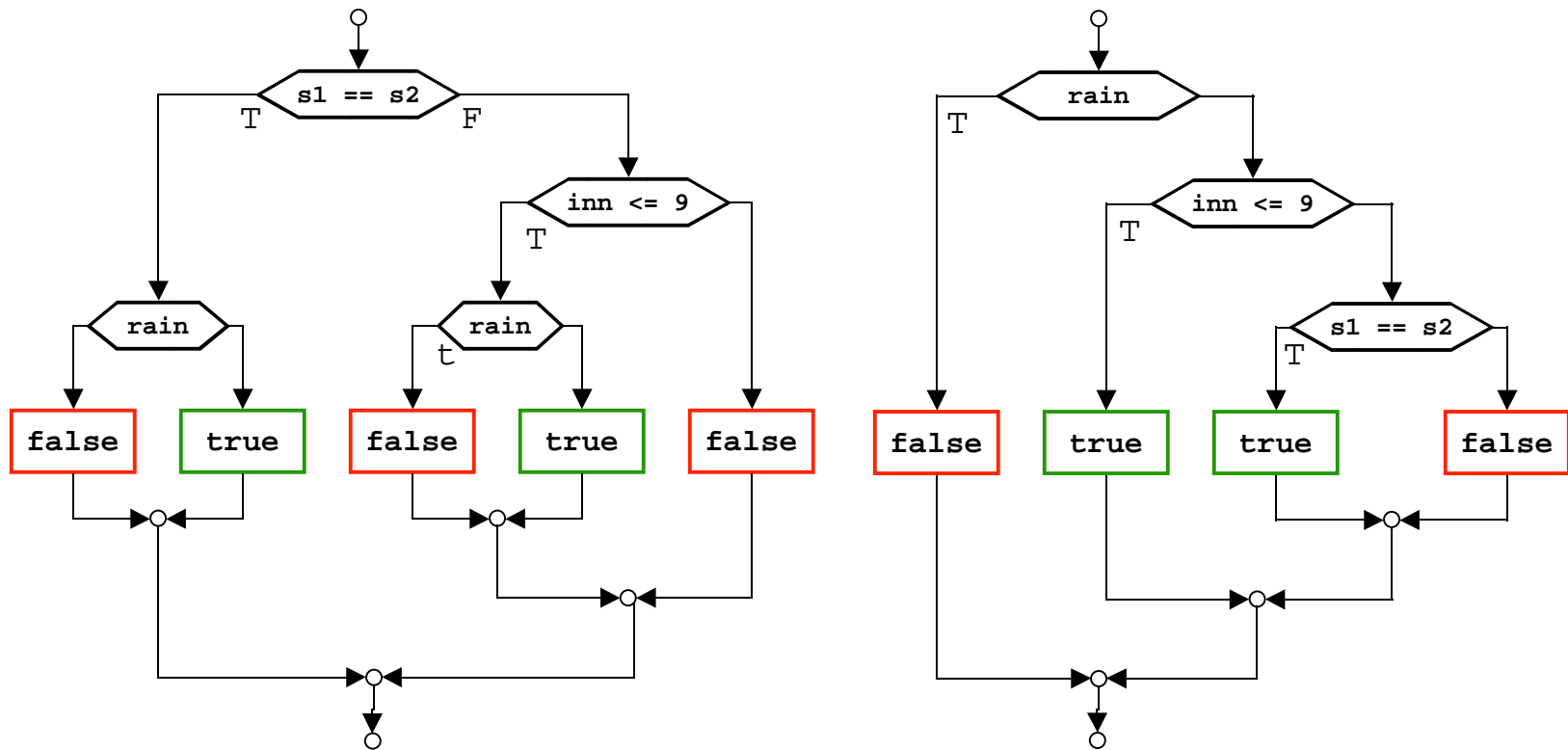


Play Ball: The condition for continuing to play a ball game can be done in many ways:

- a. In English: play when the score is tied or inning less than or equal to 9 and it's not raining.
- b. In Symbolic logic:

```
playBall = ( (score1 == score2) || (inning <= 9) ) && ! rain;
```

- c. As a control flow chart: two of them; more are possible.



There are many ways to do anything; there is a **tradeoff** between logic and program structure.

How many tests are needed to prove that this works properly? 3, 4, 5, 7, 8, 12, 16, more?

Non-Equality of Real Values

Real values (called floats and doubles) should not be compared for equality by the relation "=". This is because real values seem to be very precise, but actually may be approximate. For example, the real value 0.2 when converted to binary computer form is:

```
0.2 = 0.001100110011 ... 0011 ... .. forever
```

which continue repeating forever, unending. In practice it is chopped off (often after 32 or 64 bits), and is sufficiently accurate for most needs, but may ultimately cause problems.

Comparison between two values, say a, b, can however be made by checking if the difference between them is less than a given small value, such as 0.1 as shown below.

```
isClose = ((a-b) < 0.1) || ((b-a) < 0.1); //within a tenth
```

For closer comparison we can decrease the size of the difference:

```
isVeryClose = ((a-b) < 0.001) || ((b-a) < 0.001); //within a thousandth
```

Also, we can use an absolute value function (from Math class) and specify a "tolerance" of error:

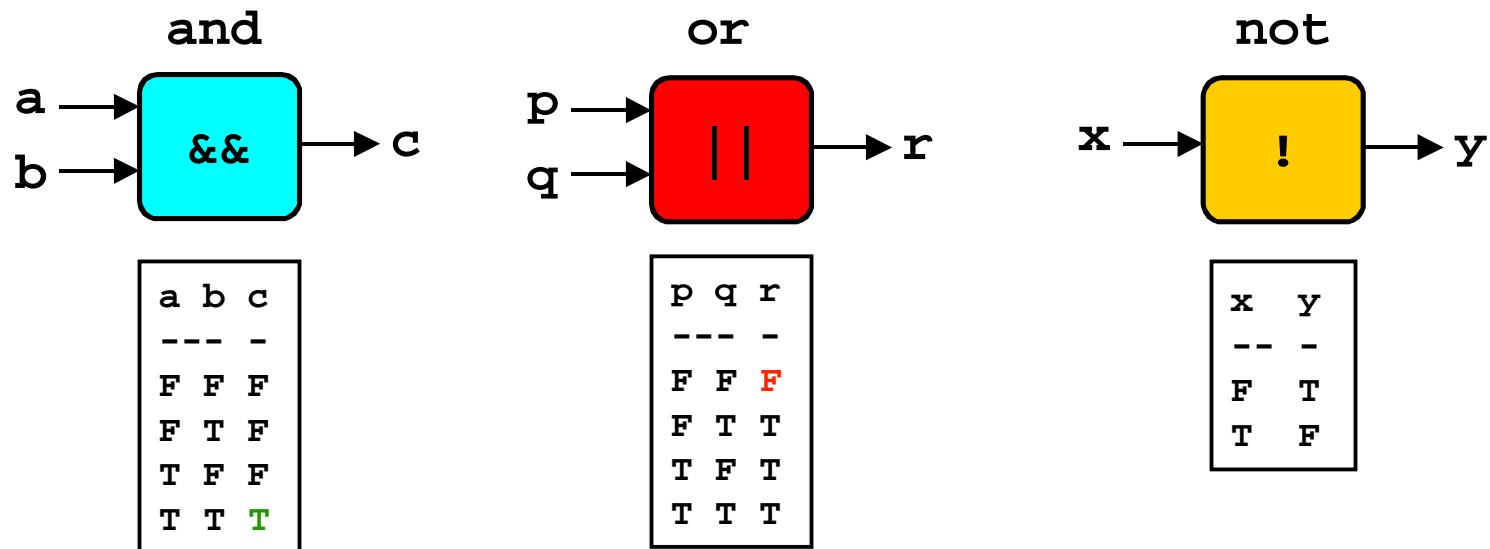
```
TOLERANCE = 0.00001; //Whatever small constant
```

```
isWithinToleranace = (Math.abs(a-b) < TOLERANCE);
```

Logic: Another View: Data Flow View

Behavior of logical operators can be shown by data flow diagrams as given below.

For example, a true value flows out of an And only if a true value flows into both inputs. Similarly, a true value flows out of an Or if a true value flows into either or both inputs.



Two-dimensional tables are an equivalent way to define some logical operators as follows.

&&	
	F T
F	F F
T	F T

	F T
F	F T
T	T T

Illogic -- Looks good

Illogical conditions or statements look and sound logical, and often have a reasonable meaning to human beings, but they are not of the proper form for computing.

For example, an "illogical" statement often written in mathematics is:

$$x < y < z$$

which humans easily interpret as the two combined relations ($x < y$) and also ($y < z$). This is written "logically" to machines as:

$$(x < y) \ \&\& \ (y < z)$$

Conversion of other illogical, but meaningful conditions for humans follow. First, look at the illogic at the left and try to convert it before looking at the right. Assume that a, b, and c are boxes of integer type.

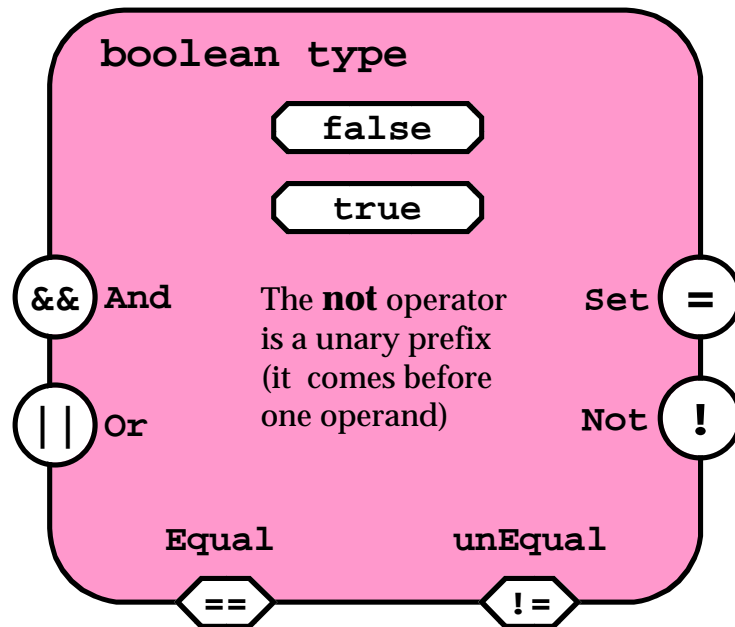
a and b < 7	converts to	(a < 7) && (b < 7)
a > b or c	converts to	(a > b) (a > c)
a <= b and c	converts to	(a <= c) && (b <= c)
a == b == c	converts to	(a == b) && (b == c)
a == b and c	converts to and also to	(a == b) && (a == c) (a == b) && (b == c)
a != b or c	converts to and also to	(a != b) && (a != c) !((a == b) (a == c))

Programming with Logic

The logical system, involving the boolean type is shown as a class diagram below. This system includes the boolean values (true and false,) three operations (&&, ||, !), assignment (=) and comparisons (== and !=). Notice that the other comparative relations (<, >, <=, >=) are not defined for this system; one truth value is not larger than another.

Other methods include input, output and conversion from boolean to string.

Logical System



Some Boolean Methods using JJS

```
JJS.inputBoolean ()  
  
JJS.outputBoolean (b);  
JJS.outputLnBoolean (b);  
  
JJS.booleanToString (b)
```

Other boolean operators include **Xor** and (not in Java) : **Nand, Nor, Inhibit,** and others.

Logic Code

Code follows showing a simple check of the And operation.
Three of the four possible cases are shown as output;
You do the fourth one; what is it?

```
// Name An Onymous
// Does logical AND
// Shows boolean code

boolean first, second, both;

JJS.outputLnString ("true or false? ");

JJS.outputLnString ("Is first true? ");
first = JJS.inputBoolean ();
JJS.outputLnBoolean (first);

JJS.outputLnString ("Is second true? ");
second = JJS.inputBoolean ();
JJS.outputLnBoolean (second);

JJS.outputLnString ("Both are true is ");

both = first && second;

JJS.outputLnBoolean (both);
```

```
true or false?
Is first true?
true
Is second true?
true
Both are true is
true

true or false?
Is first true?
false
Is second true?
true
Both are true is
false

true or false?
Is first true?
false
Is second true?
false
Both are true is
false
```

Logic DisProof

Code follows showing a check of the "nonlinearity" of the And operation.

```
!(first && second) == (! first) && (! second) //NOT SO
```

```
// Name An Onymous
// Does prove inequality
// Shows non linearity of And

boolean first, second, left, right, equal;

JJS.outputLnString ("true or false? ");

JJS.outputLnString ("Is first true? ");
first = JJS.inputBoolean();
JJS.outputLnBoolean (first);

JJS.outputLnString ("Is second true? ");
second = JJS.inputBoolean();
JJS.outputLnBoolean (second);

left = ! (first && second);
right = (!first) && (!second);

equal = (left == right);

JJS.outputLnString ("Equality is ");
JJS.outputLnBoolean (equal);
```

```
true or false?
Is first true?
false
Is second true?
false
Equality is
true
```

```
true or false?
Is first true?
false
Is second true?
true
Equality is
false
```

Disproving a property such as this "linearity"
 $f(a + b) = f(a) + f(b)$
can be done by showing just one counter-example (the second case above)

```

// Name An Onymous
// Does Majority of three
// Shows boolean evaluation

boolean first, second, third, majority;

JJS.outputLnString ("true or false? ");

JJS.outputLnString ("Is first true? ");
first = JJS.inputBoolean ();
JJS.outputLnBoolean (first);

JJS.outputLnString ("Is second true? ");
second = JJS.inputBoolean ();
JJS.outputLnBoolean (second);

JJS.outputLnString ("Is third true? " );
third = JJS.inputBoolean ();
JJS.outputLnBoolean (third);

majority = first && second ||
           first && third ||
           second && third ;

JJS.outputString ("The majority is " );
JJS.outputLnBoolean (majority);

```

```

true or false?
Is first true?
true
Is second true?
true
Is third true?
true
The majority is
true

true or false?
Is first true?
false
Is second true?
false
Is third true?
false
The majority is
false

true or false?
Is first true?
true
Is second true?
true
Is third true?
false
The majority is
true

```

There are 5 other cases;
what are they?

DeMorgan's Law is a logical equality which relates the Not of a combined logical expression. It has the two following forms.

```
!(p && q) = !p || !q      // not (p And q) = (not p) Or (not q)
!(p || q) = !p && !q     // not (p Or q) = (not p) And (not q)
```

Notice especially that the Not of the And of two truth values is equal to the Or of the Not of the two truth values; it is not equal to the And of the two Nots. In other words the Not is distributed over the And, but the And changes to an Or. Similarly the Not of Ors is the And of the Nots.

These deMorgan's laws are often used to complement a condition; sometimes the complement is more convenient to use or read.

Examples of complements follow.

```
!(old && rich) = ! old || !rich
              = young || poor
```

```
percentInRange = ! percentOutOfRange
               = ! ( (percent < 0) || (percent > 100) )
               = ! (percent < 0) && ! (percent > 100)
               = (percent >= 0) && (percent <= 100)
```

Parentheses are not required around the inner conditions, but it is useful to include them for clarity. For example, the last above example could also be written as:

```
percent >= 0 && percent <= 100
```

Proof using Truth Tables

Logical proofs involve testing all possible cases; which are often a small and finite number.

For example to prove the deMorgan Rule:

$$\neg(p \ \&\& \ q) \ == \ \neg p \ || \ \neg q$$

involves only 4 cases of p,q: FF, FT, TF, TT.

For the first case: p=F and q=F the left side becomes:

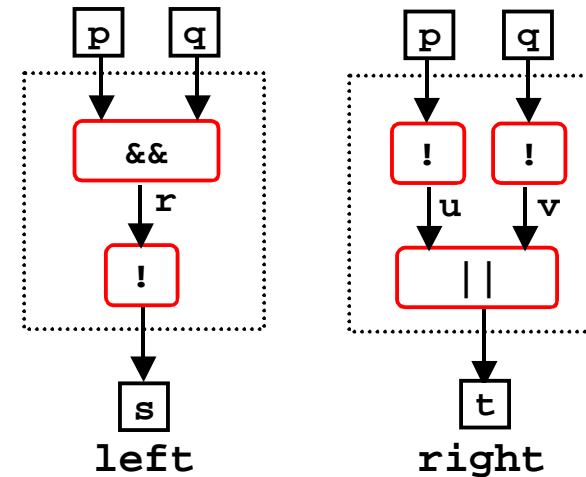
$$\neg(p \ \&\& \ q) = \neg(F \ \&\& \ F) = \neg(F) = T$$

and the right side evaluates to:

$$\neg p \ || \ \neg q = \neg F \ || \ \neg F = T \ || \ T = T$$

Similarly for the second case: p = F, q = T
both sides evaluate to the same result, which is T.

Row-by-row evaluation of all 4 cases is summarized on the given truth table, which shows the same result for all cases, so proving this DeMorgan's law.



p	q	left	right
F	F	T	T
F	T	T	T
T	F	T	T
T	T	F	F

Column-by-Column proof is also possible, as shown in the given table:

p	q	!(p && q)	==	!p !q
F	F	T	F	T
F	T	T	F	T
T	F	T	F	T
T	T	F	T	F

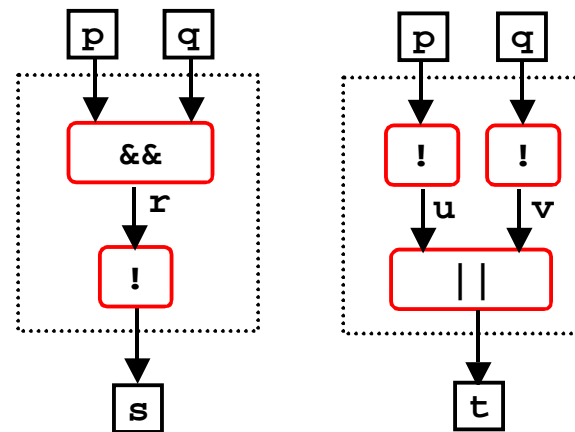
1	2	4	3	8	5	7	6
---	---	---	---	---	---	---	---

Numbers under the columns show the order in which the columns are done. The last column 8 is most important, showing the equality in all the cases.

Another proof follows; it renames some interior points (r, u, v) and is more graphical rather than algebraic. It may seem simpler in some ways.

Proof of DeMorgan's Law

p	q	r	s	u	v	t
F	F	F	T	T	T	T
F	T	F	T	T	F	T
T	F	F	T	F	T	T
T	T	T	F	F	F	F



Three logical conditions in a formula need 8 rows. For example, one unusual property of distribution follows; it shows that the common boolean box "a" is factored out of two terms. It is similar to the ordinary arithmetic formula:

$$a*b + a*c = a*(b + c)$$

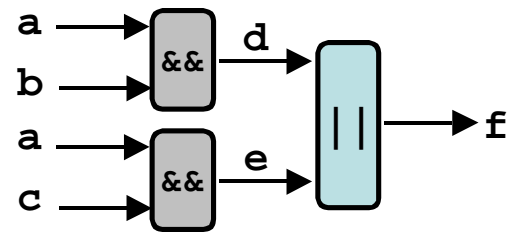
Distribution: logical factoring

$$(a \ \&\& \ b) \ || \ (a \ \&\& \ c) = a \ \&\& \ (b \ || \ c)$$

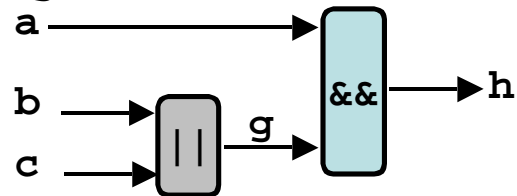
d f e i h g

a	b	c	d	e	f	g	h	i
F	F	F	F	F	F	F	F	T
F	F	T	F	F	F	T	F	T
F	T	F	F	F	F	T	F	T
F	T	T	F	F	F	T	F	T
T	F	F	F	F	F	F	F	T
T	F	T	F	T	T	T	T	T
T	T	F	T	F	T	T	T	T
T	T	T	T	T	T	T	T	T

Left Hand Side



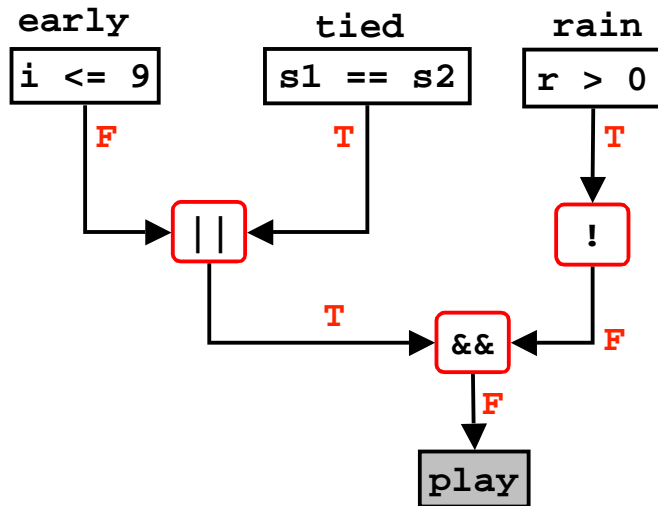
Right Hand Side



The logical data flow diagrams show that these two equivalent formulas correspond to two very different structures; one can be substituted for another.

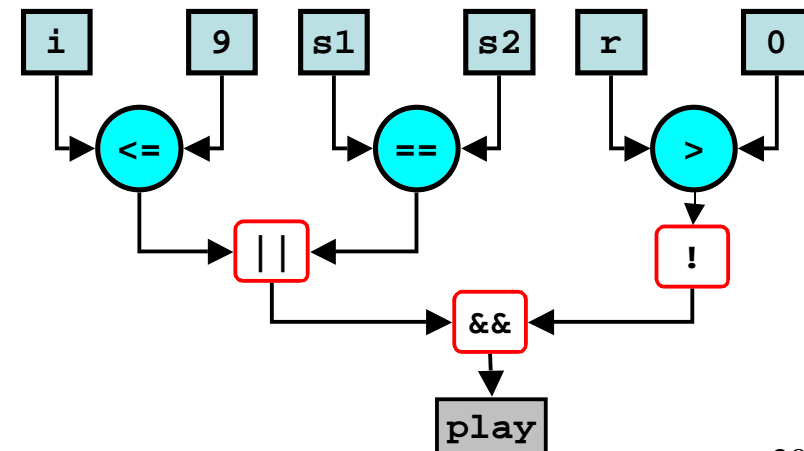
PlayBall is a condition involving 3 boolean boxes, and 8 possible combinations of the 3 conditions shown. One condition (F,F,T) is emphasized; others are equally important.

Play = (early || tied) & ! rain



early (i <= 9)	tied (s1 == s2)	rain (r > 0)	play
F	F	F	F
F	F	T	F
F	T	F	T
F	T	T	F
T	F	F	T
T	F	T	F
T	T	F	T
T	T	T	F

PlayBall can be specified in more detail, by including the numeric relational operators as shown in the data flow diagram at the right.



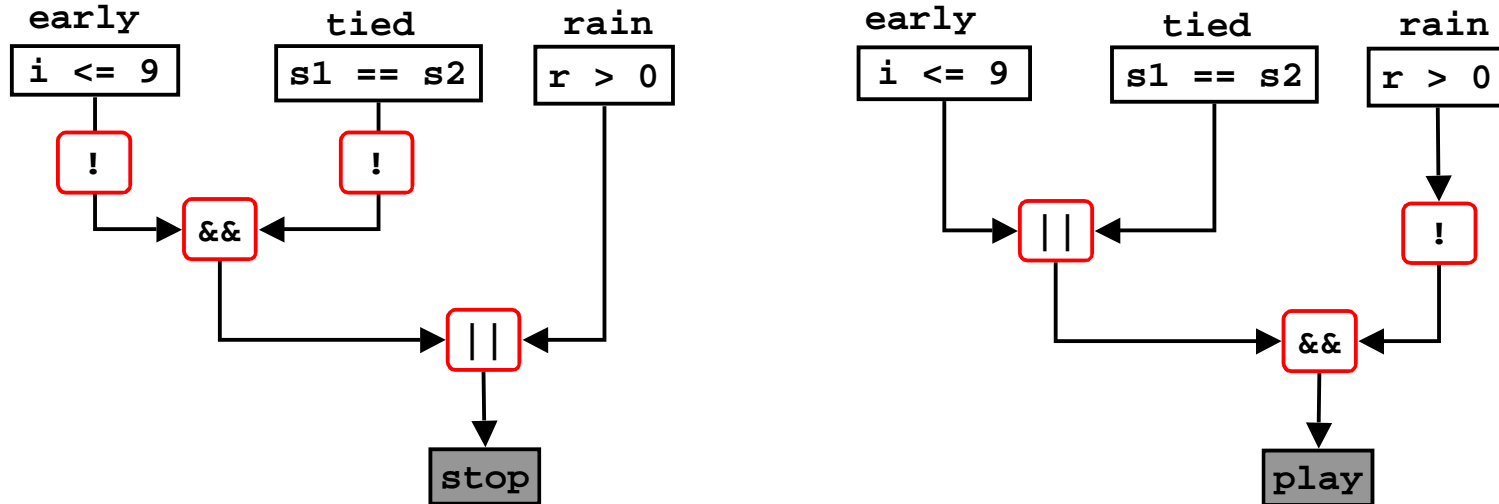
stopPlay is the complement (or Not) of playBall ;
it can be created using DeMorgan's law algebraically as follows.

```

stopPlay = ! playBall
          = ! ( (inning <= 9) || (score1 == score2) ) && ! rain )
          = ! ( (inning <= 9) || (score1 == score2) ) || ! ( !rain)
          = ! (inning <= 9) && !(score1 == score2) || rain
          = (inning > 9) && (score1 != score2) || rain

```

The data flow diagram showing this stopPlay logic follows, at the left
along with the previous playBall logic, at the right.



```

// Name An Onymous
// Does playBall algorithm
// Shows boolean evaluation

boolean tied, early, rain, play;

JJS.outputLnString ("true or false? ");

JJS.outputLnString ("Is it raining? ");
rain = JJS.inputBoolean ();
JJS.outputLnBoolean (rain);

JJS.outputLnString ("Is score tied? ");
tied = JJS.inputBoolean ();
JJS.outputLnBoolean (tied);

JJS.outputLnString ("Inning <= 9 ? ");
early = JJS.inputBoolean ();
JJS.outputLnBoolean (early);

play = (tied || early) && ! rain;

JJS.outputString ("Play is ");
JJS.outputLnBoolean (play);

```

```

true or false?
Is it raining?
true
Is score tied?
true
Inning <= 9 ?
true
Play is false

true or false?
Is it raining?
true
Is score tied?
false
Inning <= 9 ?
true
Play is false

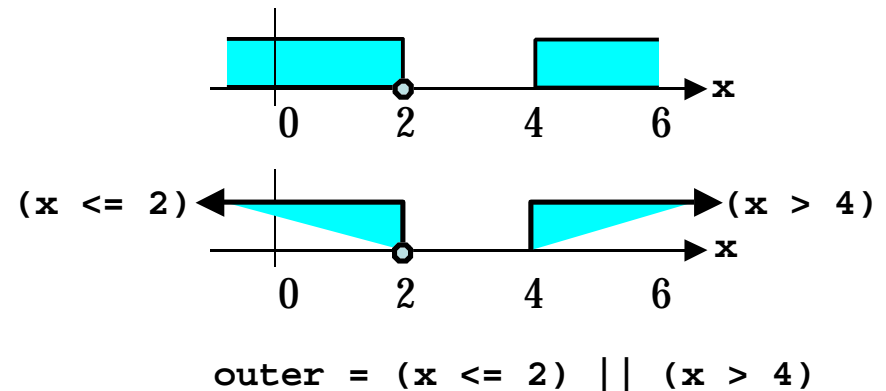
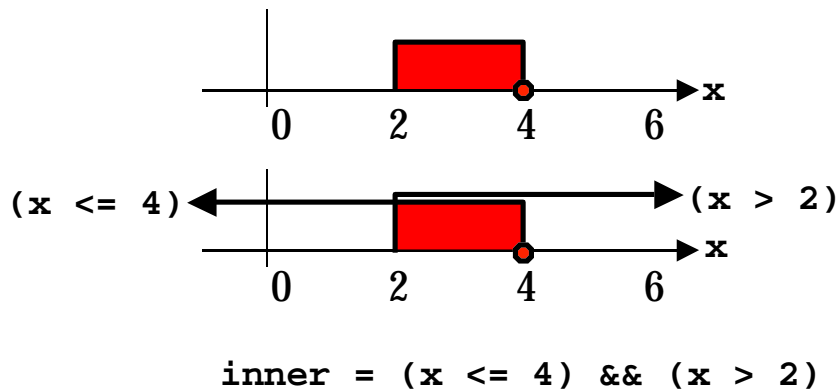
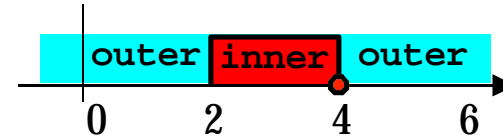
true or false?
Is it raining?
false
Is score tied?
false
Inning <= 9 ?
true
Play is true

```

There are 5 other cases;
what are they?

Complementary Ranges

A range of values on the number line can be described logically in two ways, by the "inner" values within the range, and by the "outer" values outside of it.



Ranges can also be described by logic, as shown below each of the figures.

Truth tables at the right show how the two ranges are complementary.

They also show that some combinations (on the last line) are not possible.

x	(x <= 2)	(x > 4)	inner	outer
1	true	false	true	false
3	false	false	false	true
5	false	true	true	false
?	true	true	????	?????

Bigger Complements

Prove or disprove the complementarity of the following two conditions:

`top = (x <= 30) && ((x > 20) || (x <= 10));`

`bot = (x > 10) && ((x > 30) || (x <= 20));`

There are many ways to do this; some ways are better than others.

1. Truth table (with test cases)

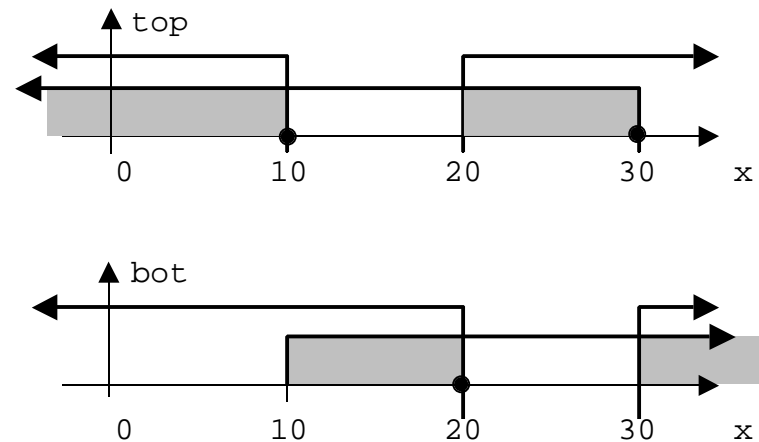
Not all cases are possible;

(x<=10)	(x<=20)	(x<=30)	x	top	bot
F	F	F	35	F	T
F	F	T	25	T	F
F	T	F	?		
F	T	T	15	F	T
T	F	F	?		
T	F	T	?		
T	T	F	?		
T	T	T	5	T	F
p	q	r		a	b

impossible

2. Graphic way

provides real insight!



3. Finally a Boolean algebra proof is not possible!

If $p = (x \leq 10)$, $q = (x \leq 20)$, $r = (x \leq 30)$ then

$r \ \&\& \ (!q \ || \ p) \ != \ !p \ \&\& \ (!r \ || \ q)$

This can be "disproved" with a truth table or by finding one case that does not hold.

Complement Code

Evaluating numeric relations can be done by code as the following.

```
// Name An Onymous
// Does evaluate complements
// Shows complementary relations

int x; // could also be double
boolean top, bot;

JJS.outputLnString ("Enter a number ");
x = JJS.inputInt ();

top = (x <= 30) && ( (x > 20) || (x <= 10) );
bot = (x > 10 ) && ( (x > 30) || (x <= 20) );

JJS.outputString ("Top is ");
JJS.outputLnBoolean (top);

JJS.outputString ("Bot is ");
JJS.outputLnBoolean (bot);
```

```
Enter a number
5
Top is true
Bot is false

Enter a number
10
Top is true
Bot is false

Enter a number
13
Top is false
Bot is true

Enter a number
20
Top is false
Bot is true
```

Simplifying Logic

Some logic can be simplified if more information is known.

For example, given three sides of a triangle, labelled s_1 , s_2 , s_3 , we can determine if it is an equilateral triangle logically as:

```
equilateral = (s1 == s2) && (s2 == s3) && (s1 == s3);
```

or more simply as:

```
equilateral = (s1 == s2) && (s1 == s3);
```

However, if it is known that the sides are in increasing order,

```
s1 <= s2 and also s2 <= s3
```

then this can be more simply written as:

```
equilateral = (s1 == s3);
```

since if the first equals the last, then the middle value must equal both.

Alternately, if a triangle is given by three angles a_1 , a_2 , a_3 then

```
equilateral = (a1 == a2) && (a1 == a3) && (a1 == 60);
```

and also

```
equilateral = (a1 == a3) && (a1 + a2 + a3 == 180);
```

as well as

```
equilateral = (a1 == 60) && (a2 == 60) && (a3 == 60);
```

Triangle Problem:

Write logical assignments to determine if three angles (ints) form:

- a. a triangle
- b. a right triangle (where one angle equals 90)
- c. an isosceles triangle (where two angles are equal)
- d. an acute triangle (where all angles are less than 90)
- e. an obtuse triangle (where one angle is more than 90)
- f. a scalene triangle (where all the angles differ in size)

Finite Quantified Logic (optional)

Quantifiers involving the words "all" or "some" are often used in logic. For example, an acute triangle has all angles less than 90 degrees; an obtuse quadrilateral has some angles larger than 90 degrees. When the number of entities (angles) is finite (such as the 3, or 4 here) then such logic simply involves a "chain" of Ands or Ors, as follows:

```
allAnglesLessThan90 = (a1 < 90) && (a2 < 90) && (a3 < 90);
```

```
someAngleMoreThan90 = (a1 > 90) || (a2 > 90) ||  
                      (a3 > 90) || (a4 > 90) ;
```

or alternately as:

```
isAcuteTriangle = allAnglesLessThan90;
```

```
isAnObtuseQuad = someAngleMoreThan90;
```

Universal quantifies (involve "all") are sometimes written as an inverted A symbol,

```
 $\forall$  i (i < 90) // For all angles i, i is less than 90
```

Existential quantifiers (involve "some") are written as an inverted E symbol.

```
 $\exists$  j (j > 90) // For some angle j, j is more than 90
```

Problems:

1. Prove the other DeMorgan result, using truth tables:

$$\!(p \mid\mid q) = !p \ \&\& \ !q$$

2. Prove or disprove the following "Big" DeMorgan Laws

$$\!((p \mid\mid q) \mid\mid r) = (!p \ \&\& \ !q) \ \&\& \ !r$$

$$\!(p \ \&\& \ q \ \&\& \ r) = !p \ \mid\mid \ !q \ \mid\mid \ !r$$

3. Draw truth tables of the two following expressions, one of which is equivalent to the "Exclusive-Or" and then convert the other to the Exclusive-Or

$$(p \ \&\& \ !q) \ \mid\mid \ (!p \ \&\& \ q)$$

$$(p \ \mid\mid \ q) \ \&\& \ !(p \ \&\& \ q)$$

4. Prove, or disprove the "dual" logical property of distribution (or factoring) corresponding to the arithmetic property, and be prepared for a surprize:

$$(a + b) * (a + c) = a + b * c$$

Problems: Poker Dice

The game of Dice Poker involves the throwing of 5 dice, each yielding a value from 1 to 6.

Some of these combinations (poker hands) occur less probably than others.

For example, `fiveOfAKind`, where all values are the same, is very rare;

This condition can be coded as:

```
fiveOfAKind = (d1 == d2) && (d2 == d3) && (d3 == d4) && (d4 == d5);
```

However, let us assume that the five dice values are in increasing order:

```
d1 <= d2 <= d3 <= d4 <= d5
```

Then the above condition can be coded more simply as:

```
fiveOfAKind = (d1 == d5); //first equals last, and rest
```

Write the code for the following conditions

(which happen to be in increasing order of probability) Hint: picture it.

- a. **fourOfAKind**, where 4 of the dice have the same value
(such as 1,1,1,1,2, or 3,5,5,5,5 or 4,4,4,4,6 etc)
- b. **fullHouse**, where 3 dice have one value and 2 have another value
(such as 11133 or 22555)
- c. **straight**, where the 5 values are in consecutive order (12345 or 23456)
- d. **threeOfAKind**, where only three dice have the same value
- e. **twoPairs**, where two dice are of one value, and two are of another value
- f. **onePair**, where only two dice have the same value

Play game of 42

Consider a game between two players which is played when both scores are less than 42, or when at least one score is at or beyond 42 but there is less than a 3 point difference in scores. Write this logical expression, draw its data flow diagram, select some good test cases. Do also the complement (reverse, or negative) of this condition.

Complements

Prove or disprove that the following conditions are opposite, (or complementary).

Hint: there are many ways; find two.

$(x \leq 30) \text{ and } ((x > 20) \text{ or } (x \leq 10))$

$(x > 10) \text{ and } ((x > 30) \text{ or } (x \leq 20))$