Logic and its importance when using Application Software

Arithmetic is about numbers. Logic is about truth. Where in arithmetic we are interested in the numerical value of an expression, in logic we are interested in the truth value of an expression. That is, is the expression **true** or **false**? In arithmetic we write and evaluate expressions that use numbers and arithmetic operators, such as 5 X 7 + 3, in logic we write and evaluate expressions that use truth values (or items that have a truth value) and logical operators such as **and**, **or** and **not**.

We believe that you are already convinced of the importance of arithmetic when using application software such as Excel. You should also appreciate that *to use arithmetic effectively and efficiently in Excel requires a sound understanding of arithmetic*. For example, the rules associated with the evaluation of an arithmetic expression, typically referred to as BODMAS or BIDMAS, influence how one writes arithmetic expressions in Excel. Thus if one wants to write an expression that sums the values in A1 and A2, and multiplies the result by the value in A3, then one would write

```
=(A1 + A2)* A3
```

recognising that without the brackets the product of A2 and A3 would be evaluated before the addition to A1.

In addition, you should appreciate that *in business one would typically only use arithmetic with the support of application software such as Excel*. In summary, *there is a mutual dependency between arithmetic and application software when used in business*. In fact there is a mutual dependency between several mathematical topics and several applications software. One such mutual dependency is between logic and spreadsheets and databases.

Before we discuss logic in detail we provide some motivation for learning. You are all now familiar with IF statements in Excel and selection queries in SQL (those that include a WHERE clause). Both of these require the use of logic. For example, in the following IF statement the expression presented in **bold** will evaluate to true or false.

```
=IF(A1 > 100, "OK", "Not OK")
```

It is the test that, if true, results in "OK" and, if false, results in "Not OK".

In the following SQL query the expression presented in **bold** will evaluate to true or false when applied to each record in the emp table.

```
Select ename, sal from emp
where sal > 30000;
```

It is the test that, if true, results in an employee name and salary being returned in the result table, and, if false, results in the employee name and salary not being returned in the result table **.**

Thus to effectively use IF statements in Excel or WHERE clauses in SQL we need to understand logic.

Truth Values, Propositions and Propositional Expressions

When you were in primary school you will have learnt about numbers, their ordering, how to represent them and how to combine them. You learnt that positive whole numbers may be represented by a series of digits (the set of whole numbers between 0 and 9) where each digit has a place value. There are therefore an unlimited number of whole numbers. That is, there is no largest whole number. Similarly, one can represent negative whole numbers as a series of digits prefixed by a -. Once again there is no smallest negative number.

You also learnt how to combine numbers using the arithmetic operators. You learnt how to write algebraic expressions such as x + y to represent the sum of any two numbers. To understand the semantics (meaning) of this expression you will have produced a table listing the evaluation of this expression for various values of x and y. For example,

x	**y**	**x + y**
1	1	2
1	2	3
2	1	3
2	2	4
2	3	5

Of course this table could go on forever since there are an infinite number of combinations of values of x and y that may be used.

We adopt a similar approach to understand the semantics of expressions with truth-valued elements. These elements are known as **propositions** (or statements). A proposition is a sentence that is either true of false. The values true and false are collectively known as the Boolean values (named after the mathematician and logician, George Boole). All of the following sentences are propositions.

London is the capital of England.

The product of two odd numbers is odd.

A square has five sides.

There are 12 players on a football team.

Note that not all of the above propositions are true. They don't have to be in order to be a proposition. They just have to be either true or false and not a matter of opinion.

Exercises Which of the following sentences is a proposition?

1. 7 is a prime number
2. Where is the LRC?
3. The module leader for BB4301 is Dan Russell
4. The sum of two odd numbers is odd
5. Turn off your mobile phone
6. Balham is in North London
7. The Great British Bake Off is the best programme on television

Before we may write expressions using propositions we need to list the operators that are available for use, and their meaning. These are: and, or, not, implication and logical equivalence. We present implication and logical equivalence later in this handout.

There is more than one way to represent and, or and not but we will simply use these names to represent the operators. The semantics (meaning) of each of these operators is presented in the table below.

Operator	**Semantics**
and	Is a binary operator that evaluates to true when both of its operands are true
or	Is a binary operator that evaluates to true when at least one of its operands are true
not	Is a unary operator that evaluates to true when its operand is false

We may represent the value(s) of expressions including these operators by using a table similar to that used to represent the value of an arithmetic expression for different inputs. These tables are called **truth tables** and unlike those for arithmetic expressions they have a finite number of rows. Below we present the truth table for **A and B** where A and B are propositions. In the table we represent true with a **T** and False with an **F**.

A	**B**	**A and B**
T	T	T
T	F	F
F	T	F
F	F	F

Here each row represents one combination of possible values for the two propositions A and B, and in the last column is the value of the expression given the proposition values. That is, when A and B are both true, **A and B** is true. For the other three combinations the value of **A and B** is false.

There are four rows because each proposition has two possible values and so there are four combinations in total.

One can use the truth table to determine the value of a particular proposition. For example, **London is the capital of England and 7 is prime** is true since each component proposition is true, and the first row of the truth table indicates that the propositional expression is true.

Exercises

1. Create the truth table for **A or B**
2. Evaluate the following propositional expressions:
    a. 2016 is a leap year and there are 26 letters In the alphabet
    b. The module code for BIA is BB4301 or the module code for BIA is BB4302
    c. The module code for BIA is BB4301 and the module code for BIA is BB4302

The truth table for **not A** is

A	**not A**
T	F
F	T

That is, London is the capital of England is true, so London is not the capital of England is false.

We can represent the values of an expression that uses two or more operators by creating a truth table for the expression. For example, the expression **A and not B** may be represented as follows:

A	**B**	**not B**	**A and not B**
T	T	F	F
T	F	T	T
F	T	F	F
F	F	T	F

Note how the final column represents the possible values of the expression. Note also how we build up the expression by using extra columns to represent sub-expressions of the expression. Note that as with arithmetic where BODMAS determines the order of evaluation of an expression there are rules applied to logic expressions. Here:

- Expressions within brackets are evaluated first
- **not** is applied next
- **and** is applied next
- then **or** is applied

Exercises

1. Create the truth table for **not A or B**
2. Create the truth table for **not (A or B)**

3. Create the truth table for **not (A and B)**
4. Create the truth table for **A and B or C**
5. Create the truth table for **not (not A)**
6. For each of the expressions above write down two sentences that matches the expression, the first of which is true and the second of which is false. For example, *15 is an odd number and Italy is in Europe* is an example of an A and B expression that is true.

Tautologies and Contradictions

A propositional expression that is always true is called a **tautology**. A propositional expression that is always false is called a **contradiction**. A tautology is indicated in a truth table by all values in the final column being true. A contradiction is indicated in a truth table by all values being false.

Exercises For each of the following propositional expressions state whether it is a tautology, contradiction or neither.

a. A and B
b. A or not A
c. A and not A
d. not A

Implication and Logical Equivalence

We often want to use logical expressions of the form 'if … then …'. For example, if today is Wednesday then tomorrow is Thursday. This combines two propositions: today is Wednesday and tomorrow is Thursday. It is true if both propositions are true or if today is Wednesday is false. We call this type of expression an implication or conditional expression and refer to the first proposition as the **antecedent** and the second as the **consequent**. Another way of stating the expression is *today is Wednesday implies tomorrow is Thursday.*

Barack Obama famously told his children before becoming president of the USA:

If I become president then I will get you a dog

When could this expression be false?

The two propositions are: I become president and I will get you a dog.

If he became president (which he did) and he got his daughters a dog (which he did) then the expression is true. If he became president but he did not get his daughters a dog then the expression is false. But what if he didn't become president?

In this case the expression is true whether he got his daughters a dog or not since he did not say what would happen if he didn't become president.

Therefore the only case when it is false is when he becomes president but does not get his daughters a dog. This may be represented in a truth table as follows where => represents implication:

A	B	A => B
T	T	T
T	F	F
F	T	T
F	F	T

The logical equivalence operator evaluates to true when both of its inputs have the same value. That is, if they are both true or both false then the expression is true. We will represent the logical equivalence operator as <=> . The following expressions are true.

2 is prime <=> 8 is not prime

London is the capital of England <=> Rome is the capital of Italy

France is an island <=> France is surrounded by sea

Here is the truth table for logical equivalence.

A	B	A <=> B
T	T	T
T	F	F
F	T	F
F	F	T

Note that two propositions are logically equivalent if they have the same truth value – they do not need to be related in any way. That is, 2 is prime is logically equivalent to London is the capital of England.

We can now complete the order of evaluation applied to logical expressions (the BODMAS for logical expressions). It is:

- Expressions within brackets are evaluated first
- **not** is applied next
- **and** is applied next
- **or** is applied
- **=>** is applied
- and finally **<=>** is applied

Exercises Evaluate the following expressions.

1. 10 is prime or 5 is prime and 4 is prime
2. 10 is prime or 5 is prime => 4 is prime
3. not 7 is prime or 5 is prime and 4 is prime
4. not (7 is prime or 5 is prime and 4 is prime)

Predicates and Predicate Expressions

A predicate is a **Boolean-valued function**. That is, it is a function that returns a Boolean value – a function that returns true or false. The following are example predicates:

- isEven that tests whether an integer is even
- salaryGreaterThan30000 that tests whether an employee's salary is greater than £30000
- departmentNumberIs2 that tests if a department number is equal to 2

Note that the last two predicates are familiar from their use in SQL SELECT statements applied to the emp table.

Note that unlike propositions one is unable to determine the truth value of each of these predicates. We cannot determine if isEven is true because we don't yet have enough information to determine its value. This can only be done when we have an integer to which isEven is applied. That is:

- isEven (10) is true
- isEven (15) is false

This is also the case with the other two example predicates. Let's view salaryGreaterThan30000 in the context of an SQL query:

```
Select ename from emp
where sal > 30000;
```

The value of sal > 30000 can only be determined when it is applied to a particular record in the table. That is, it is true for employees whose salary is greater than 30000, and false otherwise.

Exercises

Write down three sql queries that use a predicate in the where clause. Give a name to each predicate.

Predicate expressions may be created in the same way as propositional expressions. That is, we may combine several predicates by using the same logical operators that we used for propositional expressions. Now although we encourage you to use 'sensible names' to represent predicates – such as isEven and salaryGreaterThan30000 – it is not uncommon to use shorter names. Thus where A, B, P and Q may represent propositions, A(x), B(y), P(x,y) and Q(m,n) may represent predicates. Note that in each case they have at least one input (variable). For example:

- E(x): x is even
- O(y): y is odd
- G(x,y): x is greater than y
- ST(e): the salary of e is greater than 30000

Here we present the name of the predicate followed by its definition. Note how in each case the definition uses the name of the variable. These definitions require one further bit of information:

the domain of the predicate functions. That is, the set of input values. Here we present the definitions with their domains.

- E(x): x is even — Domain: set of integers
- O(y): y is odd — Domain: set of integers
- G(x,y): x is greater than y — Domain: set of integers
- ST(e): the salary of e is greater than 30000 — Domain: set of employees in emp table

We may then apply the predicate to a particular input (or inputs) from the domain and determine its value. For example,

- E(10) evaluates to true
- E(7) evaluates to false
- G(10,4) evaluates to true
- ST(Gibson) evaluates to true
- ST(March) evaluates to false

Now we may create predicate expressions by combining predicates and logical operators.

For example,

- E(x) and O(x)
- E(y) or O(y)
- E(m) and not E(m)
- not E(x) and O(x)
- not (E(x) and O(x))
- E(p) and O(q) and G(p,q)
- not ST(x)

Exercises Evaluate the following predicate expressions:

1. E(5) and O(5)
2. E(8) or O(8)
3. E(4) and not E(4)
4. not E(7) and O(7)
5. not (E(9) and O(9))
6. E(10) and O(7) and G(10,7)
7. not ST(March)
8. Write SQL queries that have:
    a. A predicate expression of the form A(x) and B(x)
    b. A predicate expression of the form A(x) or B(x)
    c. A predicate expression of the form not (A(x) and B(x))
    d. A predicate expression of the form not A(x) and B(x)
    e. A predicate expression of the form A(x) and B(x) or C(x)
    f. A predicate expression of the form A(x) and (B(x) or C(x))

Quantifiers

We may add extra functionality to predicate expressions by using quantifiers. These add information about the number of elements in the domain that are required to satisfy the predicate (make it true). There are two quantifiers: the universal quantifier $\forall$ and the existential quantifier $\exists$. The universal quantifier represented 'for all' or 'all' and the existential quantifier represents 'there exists' or 'there is'.

Here are some example predicate expressions with quantifiers. In each case we translate the expression into English.

- $\forall x\ E(x)$ All integers are even
- $\exists y\ E(y)$ There is an integer that is even
- $\forall x$ not $E(x)$ All integers are not even
- $\forall y\ (E(y)$ or $O(y))$ All integers are even or odd
- $\exists x\ \exists y\ G(x,y)$ There exists two integers one of which is greater than the other

Exercises Translate the following predicate expressions into English which use the predicates:

- F(x): x is female Domain: set of students in the class
- T(y): y is twenty years old Domain: set of students in the class
- O(p,q): p is older than q Domain: set of students in the class

a. $\forall x\ F(x)$
b. $\exists y\ T(y)$
c. $\forall x$ not $T(x)$
d. $\forall m\ (T(m)$ or not $T(m))$
e. $\exists x\ \exists y\ O(x,y)$
f. $\forall x\ \exists y\ O(x,y)$
g. $\forall s\ \forall t\ O(s,t)$
h. $\exists s\ \forall t\ O(s,t)$
i. $\exists x\ (F(x)$ and $T(x))$
j. $\exists x\ \exists y\ (F(x)$ and not $F(y)$ and $O(x,y))$