CS1113
Foundations of Computer Science II

Lecturer:

Professor Barry O'Sullivan

Office: 2.65, Western Gateway Building

email: *b.osullivan@cs.ucc.ie*

http://osullivan.ucc.ie/teaching/cs1113/

CS1113
Foundations of Computer Science II

Lectures:
Monday, 14:00, WGB 107
Wednesday 15:00, WGB G.05

Problem Solving Classes (not starting yet):
Monday 10:00-12:00, WGB G20
Monday 15:00 -17:00, WGB G20

Formal Module Description

- **Module Objective:** to develop advanced skills in the foundational techniques needed to analyse, design, implement and communicate computational problems and solutions.

- **Module Content:** Predicate logic; representing and solving computational problems with trees and graphs; analysis of simple data structures, algorithms and problem spaces.

- **Learning Outcomes:**
  - Formulate computational problems using predicate logic specifications;
  - Represent and solve computational problems with trees and graphs;
  - Analyse simple data structures and algorithms.

Informal Module Description

- **Objective:**
  - learn how to express complex problems clearly and precisely
  - learn some important abstract structures
  - learn some important algorithms
  - learn how to analyse problems and solutions

- **How are you going to achieve it?**
  - keep up with the module as you are going along
  - attend all lectures, and participate in them
  - in the problem classes, work on the problems
  - submit the assignment each week

http://osullivan.ucc.ie/teaching/cs1113/

Assessment

- **Assessment:**
  - Formal Written Examination 80 marks;
  - Continuous Assessment 20 marks (In-Class Tests 20 marks)
  - (there will be 2 in-class tests)

- **Formal Written Examination:**
  - 1 x 1½ hr(s) paper(s) to be taken in Summer 2015.

Relation to CS1112

- this module assumes you understand and are comfortable with everything covered in CS1112:
  - sets, functions, relations, propositional logic, algorithms

- you are allowed to register for CS1113 as long as you remained registered for CS1112 up to the end of the 1st term
  - formally, you do not require a pass in CS1112
  - but if you didn't get a pass, you will struggle, and you will have to work hard now to catch up

- provisional marks for CS1112 will be released in early February and final confirmed marks will be released in June

Formal Module Description

- **Module Objective:** to develop advanced skills in the foundational techniques needed to analyse, design, implement and communicate computational problems and solutions.

- **Module Content:** Predicate logic; representing and solving computational problems with trees and graphs; analysis of simple data structures, algorithms and problem spaces.

- **Learning Outcomes:**
  - Formulate computational problems using predicate logic specifications;
  - Represent and solve computational problems with trees and graphs;
  - Analyse simple data structures and algorithms.

CS1113
Introduction to Predicate Logic

Lecturer:

Professor Barry O'Sullivan

Office: 2.65, Western Gateway Building

email: *b.osullivan@cs.ucc.ie*

http://osullivan.ucc.ie/teaching/cs1113/

Introduction to Predicate Logic

A brief review of propositional logic
Why we need a more expressive logic
Predicates

Brief review of propositional logic

1. write specifications and descriptions of situations precisely and unambiguously
2. understand when one situation is implied by another (and so we can prove that an argument is valid, or that some initial assumptions lead to a conclusion)

- *propositions* are statements that are either true or false
  - propositional symbols {*p*, *q*, *r*, ...} represent propositions

- *connectives* combine propositions to get larger more complicated statements
  - $\neg p$, $p \wedge q$, $p \vee q$, $p \rightarrow q$, $p \leftrightarrow q$
    (NOT) (AND) (OR) (IF ... THEN) (IF AND ONLY IF)

Truth Tables

- We specified how to determined the truth or falsehood of a combined statement by considering the truth value of the propositions and the connective which joins them

X	$\neg X$
T	F
F	T

X	Y	$X \vee Y$
T	T	T
T	F	T
F	T	T
F	F	F

X	Y	$X \wedge Y$
T	T	T
T	F	F
F	T	F
F	F	F

X	Y	$X \rightarrow Y$
T	T	T
T	F	F
F	T	T
F	F	T

Note function representation:

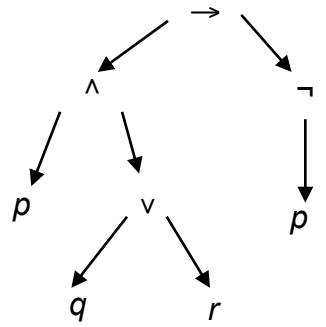
input *output*

X	Y	$X \leftrightarrow Y$
T	T	T
T	F	F
F	T	F
F	F	T

Building truth tables for larger statements

- write down a column for each propositional symbol that appears in the statement
- write a row for each possible combination of truth values for the atomic propositions
- work out the structure of the statement by building a tree
- each connective in the tree gets a new column in the table
- add the columns in order, starting at the bottom of the tree, and only adding a column for a connective when all its branches have been done
- complete the truth values for each column in turn using the basic truth tables for each connective

$(p \wedge (q \vee r)) \rightarrow \neg p$



p q r	$q \vee r$	$p \wedge (q \vee r)$	$\neg p$	$(p \wedge (q \vee r)) \rightarrow \neg p$
T T T	T	T	F	F
T T F	T	T	F	F
T F T	T	T	F	F
T F F	F	F	F	T
F T T	T	F	T	T
F T F	T	F	T	T
F F T	T	F	T	T
F F F	F	F	T	T

determines exactly those conditions in which the large statement is true, and those in which it is false

Logical Equivalence

- two statements are logically equivalent if and only if they have the same truth values in all situations

E.g.

$\neg(X \vee Y) \equiv \neg X \wedge \neg Y$
$\neg(X \wedge Y) \equiv \neg X \vee \neg Y$

$X \rightarrow Y \equiv \neg X \vee Y$

X	Y	¬X	¬Y	¬X∧¬Y	X∨Y	¬(X∨Y)
T	T	F	F	F	T	F
T	F	F	T	F	T	F
F	T	T	F	F	T	F
F	F	T	T	T	F	T

Valid Arguments

- a valid argument is a set of initial statements and a sequence of new statements, such that each new statement follows logically from the previous ones
  - if the earlier statements are true, then the new statement must also be true

- a valid argument does not state that the conclusion is true; it states that *if* the initial facts are true, *then* the conclusion must be true

- we could show a conclusion is valid by building a large truth table, consider only the rows where all initial statements are true, and show that the conclusion is also true

Rules of Inference

- We can construct a valid argument using rules of inference, which describe when we can add a new statement

$$\begin{array}{l} x \rightarrow y \\ \underline{x\quad\quad} \\ \therefore\ y \end{array}$$
Modus Ponens

$$\begin{array}{l} x \wedge y \\ \therefore\ x \end{array}$$
Simplification

$$\begin{array}{l} x \rightarrow y \\ \underline{y \rightarrow w} \\ \therefore\ x \rightarrow w \end{array}$$
Chaining

$$\begin{array}{l} x \rightarrow y \\ \underline{\neg y\quad} \\ \therefore\ \neg x \end{array}$$
Modus Tollens

$$\begin{array}{l} x \\ \therefore\ x \vee y \end{array}$$
Addition

$$\begin{array}{l} x \leftrightarrow y \\ \underline{x} \\ \therefore\ y \end{array}$$
Elimination

$$\begin{array}{l} x \vee y \\ \underline{\neg x\quad} \\ \therefore\ y \end{array}$$
Unit Resolution

$$\begin{array}{l} x \\ \underline{y} \\ \therefore\ x \wedge y \end{array}$$
Conjunction

$$\begin{array}{l} x \vee y \\ \underline{\neg y \vee w} \\ \therefore\ x \vee w \end{array}$$
Resolution

Examining a rule of inference

$$\frac{x \vee y \qquad \neg x}{\therefore\ y}$$

Unit Resolution

Whenever we have statements that match those above the line, we can add the one below the line, because it is implied: if the statements above the line are true, then the statement below the line must also be true

Proving the rule:

We are only interested in situations where

i. $x \vee y$ is true

and

ii. $\neg x$ is true

and in those cases we see that y must be true

X	Y	¬X	X∨Y
T	T	F	T
T	F	F	T
F	T	T	T
F	F	T	F

Proving a conclusion is implied by some assumptions

From the premises

1. $p \rightarrow q$
2. $\neg q \wedge r$
3. $\neg p \rightarrow s$
4. $s \rightarrow t$

derive the conclusion

t

1. $p \rightarrow q$
2. $\neg q \wedge r$
3. $\neg p \rightarrow s$
4. $s \rightarrow t$

5. $\neg q$	*from 2 by (i)*	*[x=¬q, y=r]*
6. $\neg p$	*from 1 & 5 by (ii)*	*[x=p,y=q]*
7. s	*from 3 & 6 by (iii)*	*[x=¬p, y=s]*
8. t	*from 4 & 7 by (iii)*	*[x=s, y=t]*

(i)	(ii)	(iii)
	$x \rightarrow y$	$x \rightarrow y$
$x \wedge y$	$\neg y$	x
$\therefore x$	$\therefore \neg x$	$\therefore y$

Give each new statement a number. Say what other statements it came from, using what rule, and with what substitutions.

So what have we got?

We now have a procedure for proving whether one statement follows on from the truth of a collection of other statements.

- we can work out the consequences of claims we make, or parameters we set, or systems that we design
- we can analyse arguments to see if they are valid

The procedure does **not** depend on the meaning of the basic propositions

- the procedure can be automated

All that remains is to express what we want to say in terms of basic propositions and connectives.

Example

Given the facts:

1. "if the user does not provide the correct password then the user is given access to the open area"
2. "if the user can read sensitive data then the user must have access to the restricted area"
3. "the user cannot have access to the open area at the same time as the user has access to the restricted area"
4. "the user has not provided the correct password"

show that the conclusion

"the user cannot read the sensitive data"

must follow, by representing the facts and conclusions as propositional statements and constructing a valid argument

p = the user provides the correct password
o = the user has access to the open area
r = the user has access to the restricted area
s = the user can read the sensitive data

1. $\neg p \rightarrow o$
2. $s \rightarrow r$
3. $\neg (o \wedge r)$
4. $\neg p$
5. $\neg o \vee \neg r$ *from 3, L.E*
6. o *from 4&1, M.P.*
7. $\neg r$ *from 5&6, U.R.*
8. $\neg s$ *from 2&7, M.T.*

Is there anything left to do?

Suppose we have determined the following fact:

> *the college network is secure if and only if every computer connected to the network has an active copy of FireGuard™ security suite*

We can check the status of *FireGuard™* on any computer. There are thousands of computers on the network.

How do we represent this using propositions so that
(i) we can conclude that the network is secure, or
(ii) when we detect that an individual computer has *FireGuard™* turned off, we conclude that the network is not secure?

the college network is secure if and only if every computer connected to the network has an active copy of FireGuard ™ security suite

p: the college network is secure
q: every computer connected to the network has an active copy of *FireGuard ™* security suite

$p \leftrightarrow q$

I now tell you that computer x123g has an inactive *FireGuard ™*

?

the college network is secure if and only if every computer connected to the network has an active copy of FireGuard™ security suite

Perhaps rewrite the sentence as

the college network is secure if and only if computer x1 has an active copy of FireGuard™ security suite
and x2 has an active copy of FireGuard™ security suite
and x3 has an active copy of FireGuard™ security suite
and … x3926 has an active copy of FireGuard™ security suite

I now add a new computer x3927 to the network, and it has an inactive *FireGuard™*

?

Introducing Predicates and Quantifiers

We need to expand our logical language to talk about sets of objects. We will do this using *predicates* and *quantifiers.*

Predicates are relations acting on elements of sets. Quantifiers say how many elements must satisfy the relation.

Instead of saying

- *The network is secure if and only* if *computer x1 has an active copy of FireGuard™ security suite and x2 has an active copy of FireGuard™ security suite and x3 has an active copy of FireGuard™ security suite and … x3926 has an active copy of FireGuard™ security suite*

we will go back to the original statement, and say

The network is secure if and only if

all computers on the network have an active copy of *FireGuard™*

quantifier

element in a set

predicate

Notation for predicates

Each predicate will have a name

- the name may be a word or a single capital letter

Each predicate must be applied to a specified number of objects taken from specified sets (or to variables representing those objects)

We write the objects and variables in the specified order between round brackets after the predicate name. E.g.

has_firewall(comp23)	has_firewall(x)
is_connected_to(comp23,cs_domain)	
less_than(5,9)	less_than(x, y)
P(brian)	Q(x,y)

objects

variables

Predicates and variables

In "*x*>*7*", *x* is a variable, and ">*7*" is a predicate, which is applied to a single integer (i.e. an element of Z)

If we assign the value 9 to *x*, the statement is true (9>7).
If we assign the value 1 to *x*, the statement is false (1>7).

In "x>y", *x* and *y* are variables, and ">" is a predicate, which is applied to two integers (and so ">" is a relation over ZxZ).

If we assign 5 to *x* and 1 to *y*, the statement is true (5>1).
If we assign 3 to *x* and 9 to *y*, the statement is false (3>9).

The assignment of a value to each variable in the statement turns the statement into a proposition.

Interpreting predicates

- Suppose we let P(x) stand for the statement "x > 7"

The predicate name is P, and it applies to a single object in *Z*

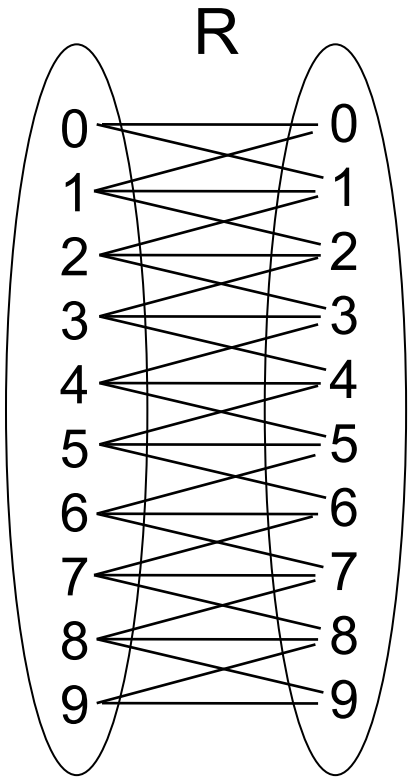
P(4) is the statement "4 > 7" (which has truth value F)
P(9) is the statement "9 > 7" (which has truth value T)
P(*x*) has no truth value

Example: what are the truth values, if any, of the following?

(i) P(1)
(ii) P(*y*)
(iii) P(12)

Predicates and Relations

Let R be the relation "is no more than 1 different from", applied to two elements of A = {0,1,2,3,4,5,6,7,8,9}



$R \subseteq AxA$

In terms of relations, for $x \in A$ and $y \in A$, if $(x,y) \in R$, we wrote xRy or $R(x,y)$

So we wrote R(4,5) and R(7,7), but we wrote ¬R(9,1).

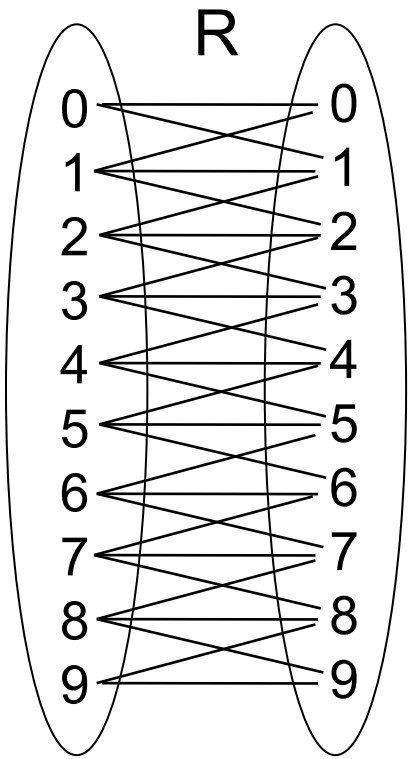
In logic, we say R(4,5) and R(7,7) are true, but R(9,1) is false.

(or we may say ¬R(9,1) is true).

Predicates and Functions

We can also think of a predicate as a function mapping from the sets to {T,F}

"is no more than 1 different from"



$R \subseteq AxA$

A	A	{T,F}
0	0	T
0	1	T
0	2	F
0	3	F
...		
1	0	T
1	1	T
1	2	T
1	3	F
...		
2	0	F
2	1	T
2	2	T
2	3	T
2	4	F
...		

Unary predicates

The truth-function representation makes it easy to think of predicates which act on a single element.

Let C be a set of computers. The predicate "has_firewall" acts on this set, and specifies which computer has a firewall.

has_firewall

C	{T,F}
comp1	T
comp2	T
comp3	F
comp4	T
comp5	F
...	

"has_firewall" is a unary predicate (or unary relation) because it acts on a single set.

Predicates: Further examples

A predicate is a relation acting on one or more sets. Each predicate is applied to a specified number of objects from specified sets, listed in a specified order.

E.g. if C is a set of national football teams , then we might have a predicate "*inWC2014*", which applies to a single object from the set C, and is true if the team qualified for the World Cup 2014 finals.

inWC2014(England) is true
inWC2014(Ireland) is false

$inWC2014 \subseteq C$

E.g. we might have a predicate "*sameGroup*", which applies to two objects from C, and is true if the two teams were in the same group in WorldCup2014

sameGroup(Italy, England) is true
sameGroup(England, Germany) is false
sameGroup(England, Scotland) is false

$sameGroup \subseteq C \times C$

Using predicates with connectives

We can combine predicates using connectives

inWC2014(Brazil) ∧ *inWC2014*(Mexico) ∧ *sameGroup*(Brazil,Mexico)

(which happens to be a true statement)

Define a predicate *beats*, which applies to two teams from C, and which is true if the first team beats the second, and a predicate *winsGroup*, which applies to one team from C, and which is true if the team wins its group.

(*beats*(Brazil,Mexico) ∧ *beats*(Brazil,Croatia) ∧ *beats*(Brazil,Cameroon)) → *winsGroup(*Brazil)

(which is also a true statement)

So what have we done?

Previously, a proposition was simple atomic statement which was either true or false, and had no internal structure.

We can now talk about

- sets of elements
- cross-products of sets of elements
- relations over sets of elements
- whether or not a tuple of elements satisfies a relation

and we can use these to create propositions.

We can also use *variables* instead of elements, to get statements which are not (yet) propositions.

Next we will define a way to talk about these statements which contain variables.

Next lecture

Quantifiers