Formal Specification
*A brief acquaintance from a Software Engineering point of view.*

In this part of the course you will be introduced to the notion of specifying system behaviour through the use of formal concepts and using formal notation. Formally describing system functions paves the way to proving the correctness (or not) of systems and allows the verification and automation of this process. It is practically impossible to find a modern critical system that has not been formally specified and has had some verification done to it.
Session Contents

- Formal specification
- Sets and comprehensive specification
- Briefly on Propositional and Predicate calculi
- Algebraic specifications
- Z-schema using sets
- Z-schema using sequences
What is Formal Specification

The application of set theory, propositional and predicate calculi, or their derivatives, to the specification of systems.
Sets

- Are collections of elements
- Are represented by standard notation
- Are manipulated by standard elementary operations
- Are entities which can interact with each other
Examples of Sets

Notation: \{ \}  
- Set of colours: \{green, blue, yellow\}  
- Set of sports: \{tennis, football, equestrian\}  
- Set of lecturers in this room: \{ernest\}  
- Empty set: \{ \} or \emptyset
Set Construction

(Comprehensive Specification of resulting sets)
• Direct *as in previous slide*
• Operations on other sets according to the form:

  The resulting set whose elements are formed by the operation on elements selected by the condition from the original set with elements from range.

Can be represented as…

\[ \{ \text{set : range} \mid \text{condition} \cdot \text{operation} \} \]

Furthermore, in notational form:

\[ \{ \text{Signature} \mid \text{Predicate} \cdot \text{Term} \} \]

\[ \{ x : X \mid P(x) \cdot E(x) \} \]
Set Construction Examples

Alternate even numbers:
\[ \{ x : N \mid x \mod 2 = 0 \cdot 2 \cdot x \} = \{0,4,8,12,16,20,…\} \]

Tens:
\[ \{ x : N \mid x \cdot 10 \cdot x \} = \{0,10,20,30,40,…\} \]

Squares of multiples of 4 (excluding zero):
\[ \{ x : Z \mid (x \mod 4 = 0) \land (x > 0) \cdot x \cdot x \} = \{16,64,144,256,…\} \]
Write 3 elements from the sets specified by the following comprehensive specifications:

\{ n : \mathbb{N} \mid n > 10 \land n < 20 \} \quad \{11, 12, 13, \ldots\}
\{ n : \mathbb{N} \mid n^3 > 10 \} \quad \{3, 4, 5, \ldots\}
\{ x, y : \mathbb{N} \mid x + y = 100 \} \quad \{(0, 100), (1, 99), (2, 98), \ldots\}
\{ x, y : \mathbb{N} \mid x + y = 5 \cdot x^2 + y^2 \} \quad \{25, 17, 13, \ldots\}

Interpret the following:

\{ m : \text{monitor} \mid \text{MonitorState}(m, \text{on}) \} \quad \{m: \text{monitor}\}
\{ f : \text{SysFiles} \mid f \in \text{DelFiles} \land f \in \text{ArcFiles} \} \quad \{f: \text{SysFiles}\}

Write the comprehensive specification of:

\{(10, 100), (11, 121), (12, 144), (13, 169), (14, 196)\}
\{ x : \mathbb{N} \mid x \geq 10 \} \quad \{x:x: \mathbb{N} \mid x \geq 10 \}
Set Operators

If more explanation than what is given here is required, operator definitions can be found in any basic math textbook or Internet sources.

Examples:

• Mosta ∈ {Maltese towns}
• London ∉ {Maltese towns}
• #{joe,veronica,mark} = 3
• {paul,richard,claire,george} ⊆ {Group B}
• IP{milan,juve} = { { }, {milan}, {juve}, {milan,juve} }
• {dobie,collie,poodle} ∪ {poodle,labrador} =
  {dobie,collie,poodle,labrador}
• {dobie,collie,poodle} ∩ {poodle,labrador} = {poodle}
• {dobie,collie,poodle} \ {poodle,labrador} = {dobie,collie}
Propositional Calculus

- Deals with logic;
- Basically consists of statements which can be true or false (the Excluded Middle Law);
- Are mutually exclusive - never true or false at the same time (the Contradiction Law);
- Is fundamental (forms the basis of all other logical frameworks);
- Is axiomatic (not subject to further proof);
- In theory, can be used to describe anything that can be stated as statements.
Proposition (and not) Examples

- Some birds can fly
- The nation of Malta is in Asia
- Dogs are mammals
- All fish live in water
- All fish live in sea water
- Mary is the only lady in our group

- Sit down.
- How are you today?
- Get my tea, please.
- What is the weather like?
Representing Propositions

Consider the following…

• 10 is greater than 8
• 8 is less than 10
• 8 < 10
• 10 > 8
• There is a positive number such that if we add it to eight the result would be ten.

All the above are one and the same proposition
Propositional Calculus Notation

Please online sources or Software Engineering textbooks containing basic formal specification sections, for a listing of basic propositional operators.
Examples to Check

True or False?

\[ \neg((P \lor Q) \Rightarrow Q) \] \text{ true}

\[ \neg((P \land Q) \lor \neg R) \iff P \] \text{ false}

\[ \neg P \land (P \lor (Q \Rightarrow P)) \] \text{ false}

\[ ((P \Rightarrow Q) \land (R \Rightarrow S) \land (P \lor R)) \Rightarrow (Q \lor S) \] \text{ false}

**Taking:**
P as true
Q as false
R as false
S as true
Contradictions and Tautologies

• A *contradiction* is a proposition that is **always false** for all possible values and variables making it up.

• A *tautology* is a proposition that is **always true** for all possible values and variables making it up.

Examples:

\[ a \land \neg a \]  contradition

\[ a \lor \neg a \]  tautology

\[ (a \land b \land c) \Rightarrow (c \Rightarrow a) \]  tautology
De Morgan’s Laws (at the heart of simplification)

\[\neg(P \land Q) \iff \neg P \lor \neg Q\]

\[\neg(P \lor Q) \iff \neg P \land \neg Q\]

The above can be summarised in Maltese and English as follows:

**Jekk mhux abjad u iswed (meħudin f’daqqa), ifisser li mhux abjad jew mhux iswed (i.e. xi wieħed minnhom m’huiex).** Mentri, jekk mhux abjad jew iswed (i.e. l-ebda wieħed minnhom m’hu), ifisser li la hu abjad u lanqas hu iswed.

**If not white and black (taken together), means not white or nor black (i.e. it’s not one or the other). While, if not white or black (i.e. it’s none of them), means it isn’t white and neither is it black.**
Consider the following text fragment describing an aspect of the behaviour of an intruder alarm system:

“The system should be considered to be ready for intruders (alert) only when it is armed and in practice alert mode. If the system is in teaching mode and in practice alert mode, then it is considered to be alert. The system should be able to be in teaching mode and in practice alert mode while still being not alert.”
Consider the following text:

The system should be considered to be ready for intruders (alert) only when it is armed and in practice alert mode.

\[ \text{alert} \iff \text{armed} \land \text{practice} \]

If the system is in teaching mode and in practice alert mode, then it is considered to be alert.

\[ \text{teaching} \land \text{practice} \implies \text{alert} \]

The system should be able to be in teaching mode and in practice alert mode while still being not alert.

\[ \text{teaching} \land \text{practice} \land \neg \text{alert} \]

Therefore…
Detecting Contradictions

From the previous analysed specification:
alert ↔ armed ∧ practice
teaching ∧ practice → alert
teaching ∧ practice ∧ ¬alert

The second and third propositions yield a contradiction:
teaching ∧ practice → alert
teaching ∧ practice ∧ ¬alert
alert ∧ ¬alert …*contradiction*!

Furthermore, the first proposition yields another contradiction:
teaching ∧ practice ∧ ¬(armed ∧ practice) *simplifies to*…
alert ∧ ¬armed …*contradiction because being alert requires being armed*!
Consider This Statement

“Fido is a dog, dogs like bones, so Fido likes bones”.

Propositional analysis of this sentence yields three propositions, namely:
Fido is a dog (propos. 1) …let’s call this “P”
Dogs like bones (propos. 2) … “Q”
Fido likes bones (propos. 3) … “R”

Can we derive “R” from “P” and “Q” using purely propositional calculus? – Naturally, no.

Therefore…
Introducing the notion of “a predicate”

Formally, predicates can be seen as direct indicators of object properties and relationships. Denoted as $P(x)$, where $P$ denotes the predicate on the term(s) represented by $x$.

• Examples of unary predicates:
  
  dog(fido) =true;
  dog(lecturer) =false. ...(?)

• Examples of $n$-ary predicates:
  
  owned(Labrador,Boxer);
  father(John,Mary);
  team(Pawlu,Bertu,Ċensu).
Introducing Quantification
(making predicate calculus)

• Places bounds on free variables (i.e. names of objects)

\[ P(x) \text{ is a unary predicate} \]

\[ ^{(1)} \exists x \cdot P(x) \quad \text{and} \quad ^{(2)} \forall x \cdot P(x) \quad \text{Produce propositions} \]

1. There exists an object ‘x’ to which the predicate ‘P(x)’ applies.
2. For all objects ‘x’, the predicate ‘P(x)’ applies.

Some examples:
\[ \exists x: \text{staff\_age} \cdot x > 50 \text{ meaning, there is staff who is older than 50;} \]
\[ \forall x: \text{names} \cdot \text{relatives}(x) \text{ meaning, the persons by these names are all relatives.} \]
Say, we wish to state that:
“Fido is a dog and dogs like bones, so Fido likes bones”.

Taking the previously defined propositions $P,Q,R$, using predicate notation (through qualification), we can state that:
$$\forall P,Q,R; P \land Q \rightarrow R$$
Predicate Calculus in Definitions

- Can be viewed as conditional statements obeying propositional behaviour with specific values.
- **Consider a triangle** - We can say the following...

For any triangle:
1. It will consist of three sides;
2. Any of its sides will be greater than zero length;
3. The sum of the length of any two of its sides will be greater than the length of the remaining side.

Therefore...
In predicate calculus form the basic properties of any triangle “T” could be written as follows (using qualification and predicate notation):

\[ T := \text{if } 1 \land 2 \land 3, \quad \text{or more formally…} \]
\[ T := \forall a, b, c \in \mathbb{R} \quad \bullet \quad P(a, b, c), \quad \text{or more specifically…} \]

Taking \( P(a, b, c) := ((a > 0) \land (b > 0) \land (c > 0)) \land ((a + b > c) \land (b + c > a) \land (a + c > b)) \)

yields…
\[ T := \forall a, b, c \in \mathbb{R} \quad \bullet \quad ((a > 0) \land (b > 0) \land (c > 0)) \land ((a + b > c) \land (b + c > a) \land (a + c > b)) \]
Consider the predicates:

\[ \text{numerically\_bigger\_than}(x,y) \]
\[ \text{are\_equal}(a,b) \]

Can be written as…

\[ x > y \]
\[ a = b \]
Quantification Examples

\[ \exists \ i : 1..10 \cdot i^2 = 64 \]
\[ \exists \ proc : \text{processors} \cdot \text{ProcessorState}(proc, \text{active}) \]
\[ \exists \ i : \mathbb{N} \cdot i > 10 \ \text{MonitorTemp} = i \]
\[ \exists \ m : \text{AllocatedMonitors} \cdot \text{MonState}(m, \text{ready}) \]
\[ \exists \ i : 1..100; \ m : \text{AllocatedMonitors} \cdot \text{activity}(m, \text{functioning}) \land \text{AmbientTemp} = i \]
\[ \exists \ r : \text{CurrentReactors}; \ m : \text{AllocatedMonitors} \cdot \text{MonState}(m, \text{functioning}) \land \text{connected}(r, m) \]
In Natural Language (loosely adopted from Behforooz, A.)

∀x, y, z ∨ x > y ∧ y > z ⇒ x > z

∃x ∃x > 10 ∨ x + y < 100

∀x, y ∈ N → x + y ∈ N

∃x, y ∈ {1, 2, 3, 4} ∃x + y ∈ {1, 2, 3, 4}

∀x, y ∈ {1, 2, 3, 4} ∨ x > y ⇒ x − y ∈ {1, 2, 3, 4}

~ (p ∧ q) ⇐⇒ ~ p ∨ ~ q

x > y ⇐⇒ x − y > 0

x + y > 0 ⇐⇒ x > 0 ∧ y > 0

There exist x and y from the set {1,2,3,4} such that the sum of x and y is also a member of the set {1,2,3,4}

For all values x and y from the set {1,2,3,4} for which x is greater than y, the difference between x and y is also an element of the set {1,2,3,4}

The complement (negation of) two logical AND-ed values is the same as the complement of each OR-ed value

The numeric value x is greater than y if and only if the difference between x and y is positive

If the sum of x and y is positive, it cannot be concluded that both x and y are positive
Algebraic Specifications

- A specification technique mainly used for abstract data types
- Based on a strong mathematical foundation – namely algebra
- Have been in use for relatively long periods of time
- Are universal in their application
- Employ fundamental principles
Building Algebraic Specifications

- Clearly comprehend the system to model
- Determine the operations necessary for the system you have in mind
- Specify the relationship between the system's operations
- Write the specification down according to adopted standard (a popular standard is the Common Algebraic Specification Language – CASL)
**Type:** <resulting type>  
**Imports:** <what it uses/assumes>  

---

**Signatures:**  
<a list of the types that result from every operation specified within the schema>

---

**Axioms:**  
<a list of the resulting value from each operation specified in the schema>
**Type:** queue(Z)  
**Imports:** Boolean

---

**Signatures:**
- Create $\rightarrow$ queue(Z)  
- AddItem(Z,queue(Z)) $\rightarrow$ queue(Z)  
- RemItem(Z,queue(Z)) $\rightarrow$ queue(Z)  
- GetFirst(queue(Z)) $\rightarrow$ Z  
- GetLast(queue(Z)) $\rightarrow$ Z  
- isEmpty(queue(Z)) $\rightarrow$ Boolean

---

**Axioms:**
- isEmpty(Create) = true  
- isEmpty(AddItem(z,q)) = false  
- RemItem(AddItem(z,q),q) = q  
- GetLast(AddItem(z,q)) = z  
- GetFirst(AddItem(z,Create)) = z
Queue Operations Example (1/2) (loosely adopted from Pressman, R. S.)

A message queue:

Operations:
- add an item [AddItem]
- remove an item [RemItem]
- check if empty [IsEmpty]
- get first queue item [GetFirst]
- get last item [GetLast]
- create a new queue [Create]
Various example will be discussed during lectures.
The Z-Specification Language

- Attempts to place a notational framework on formal system specification
- Based on set theory
- Is model-based (relies on well understood mathematical entities and their relationship)
- Equally used to model (specify) state as well as operations on states
Some Basic Z-Schema Examples (1/2)

\[
a : N \\
b : \{7, 1, 3, 24\}
\]

\[a \in b\]

\(a\) is a natural number and \(b\) is a set formed of natural numbers as shown. \(a\) is contained in \(b\).

Note: Generically, to indicate “a set of”, the notation “\(P\) (with a hollow stem)”. Example “\(b: PN\)”
Linear equivalent would be: \([a:N; b:\{7,1,3,24\} \mid a \in b]\)
Some Basic Z-Schema Examples (2/2)

\[
\begin{align*}
\text{a, b : N} \\
\text{c : PN} \\
\text{a \in c} \\
\text{b \in c}
\end{align*}
\]

Is equivalent to…

\[
\begin{align*}
\text{a, b : N; c : PN} \mid a \in c \land b \in c
\end{align*}
\]
Naming Schemas

\[ Mon\text{Condition} \]
\[ MonNo : N \]
\[ AvailableMonitors : PN \]
\[ MonNo \in AvailableMonitors \]

Or…

\[ Mon\text{Condition} \overset{\text{def}}{=} \]
\[ [MonNo : N; AvailableMonitors : PN | MonNo \in AvailableMonitors] \]
• **Delta**
  - Denoted by the Greek literal ($\Delta$)
  - Used to extend the schema components to indicate update operations, i.e. changes in state variables (updating operations).

• **“Xi”**
  - Denoted by the Greek literal ($\Xi$)
  - Used to indicate that stored data is not affected, i.e. enquiry operations.
Specify a system which will keep track of students who have handed in Assignments. There are clearly three sets involved…

- **Class** (all the students in the class)
- **HandedIn** (all the students in the class who have handed in their assignment)
- **NotHandedIn** (all the students in the class who have not handed in their assignment)

\[ \Delta \text{Assignment} \]

\[
\begin{align*}
\text{Class}, \text{HandedIn, NotHandedIn} : & \quad P \text{ STUDENTS} \\
\text{Class}', \text{HandedIn}', \text{NotHandedIn}' : & \quad P \text{ STUDENTS} \\
\text{HandedIn} \cup \text{NotHandedIn} = & \quad \text{Class} \\
\text{HandedIn} \cap \text{NotHandedIn} = & \quad \emptyset \\
\text{HandedIn}' \cup \text{NotHandedIn}' = & \quad \text{Class}' \\
\text{HandedIn}' \cap \text{NotHandedIn}' = & \quad \emptyset
\end{align*}
\]
Model the handing in of a student assignment:

\[
\begin{align*}
\text{HandIn} \\
\text{stud} ? : \text{STUDENTS} \\
\Delta \text{Assignment} \\
\text{Stud} ? \in \text{NotHandedIn} \\
\text{NotHandedIn}' = \text{NotHandedIn} \setminus \{\text{Stud} ?\} \\
\text{HandedIn}' = \text{HandedIn} \cup \{\text{Stud} ?\} \\
\text{Class}' = \text{Class}
\end{align*}
\]
Use of a “Xi” Schema (based on previous example)

Model a query for the number of students who have handed in:

\[ \exists \text{Assignment} \]
\[
\text{Class, HandedIn, NotHandedIn} : \text{P STUDENTS} \\
\text{Class', HandedIn', NotHandedIn'} : \text{P STUDENTS}
\]
\[
\text{NotHandedIn'} = \text{NotHandedIn} \\
\text{HandedIn'} = \text{HandedIn} \\
\text{Class'} = \text{Class}
\]

Therefore:

\[ \text{AssignQuery} \overset{\text{def}}{=} \]
\[
[HandedIn! : N ; \exists \text{Assignment} \mid HandedIn! = \#\text{HandedIn}]\]
Schema Inclusion

A simple example of this will be presented during lectures.
Another Schema Inclusion Example (1/2)

FileStatus

\[\text{AllFiles, FreeFile, FilesInUse} : \mathcal{P} \text{FILES} \]
\[\text{File} : \text{FILES} \]
\[\text{RegisteredUsers} : \mathcal{P} \text{NAMES} \]
\[\text{User} : \text{NAMES} \]

User ∈ RegisteredUsers
File ∈ AllFiles
AllFiles = FreeFiles ∪ FilesInUse
FreeFile = AllFiles \ FilesInUse
FreeFiles ∩ FilesInUse = \emptyset
FreeFile ∈ FilesInUse

UserStatus

FileStatus
InvalidUsers : \mathcal{P} \text{NAMES}

User ∈ RegisteredUsers
User \notin InvalidUsers
RegisteredUsers ∩ InvalidUsers = \emptyset
Results in the following schema…

\[
\text{FileAndUserStatus} \quad \text{______________________________}
\]

\[
\begin{align*}
\text{AllFiles, FreeFile, FilesInUse} & : P \ \text{FILES} \\
\text{File} & : \text{FILES} \\
\text{RegisteredUser, InvalidUser} & : P \ \text{NAMES} \\
\text{User} & : \text{NAMES} \\
\text{User} & \in \text{RegisteredUsers} \\
\text{User} & \notin \text{InvalidUsers} \\
\text{RegisteredUsers} \cap \text{InvalidUsers} & = \emptyset \\
\text{File} & \in \text{AllFiles} \\
\text{AllFiles} & = \text{FreeFiles} \cup \text{FilesInUse} \\
\text{FreeFile} & = \text{AllFiles} \setminus \text{FilesInUse} \\
\text{FreeFiles} \cap \text{FilesInUse} & = \emptyset \\
\text{FreeFile} & \notin \text{FilesInUse}
\end{align*}
\]
Another Schema Inclusion Example (taken from Ince)

\[\text{SetInv}\]
Upper, lower \(\subseteq\) PN
MaxSize: \(N\)

\[#upper + \#lower \leq \text{MaxSize}\]

\[\text{MidInv}\]
middle \(\subseteq\) upper \(\cup\) lower

\[\text{The above schemas result in…}\]

\[\text{MidInv}\]
Middle \(\subseteq\) PN
Upper, lower \(\subseteq\) PN
MaxSize : \(N\)

\[\text{middle} \subseteq \text{upper} \cup \text{lower}\]

\[#upper + \#lower \leq \text{MaxSize}\]
Sequences

- Are not sets
- Can be viewed as collections with predefined constraints
- Exist in different forms
- Can have operations applied to them
- Widespread use in computer systems
Types of Sequences

- Normal (including empty)
  \( seq \)
- Non-empty
  \( seq_1 \)
- Injective (not containing duplicates)
  \( iseq \)

Therefore…
Formal Sequence Definitions

• Definitions…

\[ \text{seq } T = = \{ f : \mathbb{N} \rightarrow T \mid \text{dom } f = 1 .. \#f \} \]

\[ \text{seq}_1 T = = \{ f : \text{seq } T \mid \#f > 0 \} \]

\[ \text{iseq } T = = \text{seq } T \cap (\mathbb{N} \times T) \]

• Some examples…

E.g. of (seq N) is \{1 \ 3, 2 \ 9, 3 \ 9, 4 \ 11\}
written as \langle3,9,9,11\rangle

E.g. of (iseq files) is \{1 \ UpdateFile, 2 \ LogFile, 3 \ TaxFile\}
written as \langle UpdateFile,LogFile,TaxFile\rangle
Examples of these will be presented during lectures.
**Sequence (in Z) Example**

---

**FileQueue**

\[
\text{InQueue}, \text{OutQueue} : \text{seq Files}
\]

\[
\#\text{InQueue} < \#\text{OutQueue}
\]

---

**Rentals**

\[
\text{Pending}, \text{Overdue} : \text{seq ID}
\]

\[
\text{MostOverdue!}, \text{SoonToBeOverdue!} : \text{ID}
\]

\[
\text{MostOverdue!} = \text{head Overdue}
\]

\[
\text{SoonToBeOverdue!} = \text{head Pending}
\]
Heads n’ Tails

• Formally defining the “head”, “last”, “tail”, and “front” sequence operators using a Z-schema.

\[
\begin{align*}
[\text{SeqOps}] & \\
\forall s : \text{seq} \rightarrow \text{SeqOps} \bullet \\
\text{head } s &= s(1) \land \\
\text{last } s &= s(#s) \land \\
\text{tail } s &= (\lambda n : 1 .. #s-1 \bullet s(n+1)) \sqcap (\{1\} \sqcap s) \land \\
\text{front } s &= (n : 1 .. #s-1 \bullet s(n))
\end{align*}
\]
Summary

• Formal approaches
• Sets, propositions and predicates
• Algebraic specifications
• Z-Schemas