COMP201 Software Engineering I
Lecture 12 – Formal Specification

Lecturer: T. Carroll

Email: Thomas.carroll2@Liverpool.ac.uk

Office: G.14

See vital for all notes

Coming Up…

Coming Up...

- **To explain** why formal specification techniques help discover problems in system requirements
- **To describe the use of:**
  - algebraic techniques (for interface specification) and
  - model-based techniques (for behavioural specification)
- **To introduce** Abstract State Machine Model (ASML)

Formal Methods

- **Formal specification** is part of a more general collection of techniques that are known as ‘formal methods’
- These are all based on the mathematical representation and analysis of software
- **Formal methods** include:
  - Formal specification
  - Specification analysis and proof
  - Transformational development
  - Program verification
- See COMP 313 : Formal Methods for more…

The Reality of Formal Methods

- Software first developed with Assembly Language
- Limited understanding of software testing and testing tools
- Modern software development has many ways to make high quality software
- Therefore: formal methods not commonly used
  - The most acceptable techniques are approaches like programming by contract (e.g. Eiffel method)

Acceptance of Formal Methods

- Formal methods have not become mainstream software development techniques as was once predicted
- Other software engineering techniques have been successful at increasing system quality.
- Market changes have made time-to-market rather than software with a low error count the key factor.
- The scope of formal methods is limited.
- Formal methods are hard to scale up to large systems

Use of Formal Methods

- Their principal benefits are in reducing the number of errors in systems so their main area of applicability is critical systems:
  - Air traffic control information systems,
  - Railway signalling systems
  - Spacecraft systems
  - Medical control systems

- In this area, the use of formal methods is most likely to be cost-effective

Specification in the Software Process

- **Specification** and design are inextricably mixed.
- **Architectural design** is essential to structure a specification.
- **Formal specifications** are expressed in a mathematical notation with precisely defined vocabulary, syntax and semantics.

Specification Techniques

- **Algebraic approach**
  - The system is specified in terms of its operations and their relationships
- **Model-based approach**
  - The system is specified in terms of a state model that is constructed using mathematical constructs such as sets and sequences.
  - Operations are defined by modifications to the system's state

Use of Formal Specification

- **Formal specification involves investing more effort in the early phases** of software development

  This reduces requirements errors as it forces a detailed analysis of the requirements

- **Incompleteness** and **inconsistencies** can be discovered and resolved.

  Hence, savings are made as the amount of rework due to requirements problems is reduced

Formal Specification

- The system requirements and design are expressed in detail
- This reduces ambiguity
- Requirements are carefully analysed and refined before implementation
- A large benefit of formal specification is its ability to uncover potential problems and ambiguities in the requirements.

Question Time

CLICK HERE TO GO TO POLL

Interface Specification

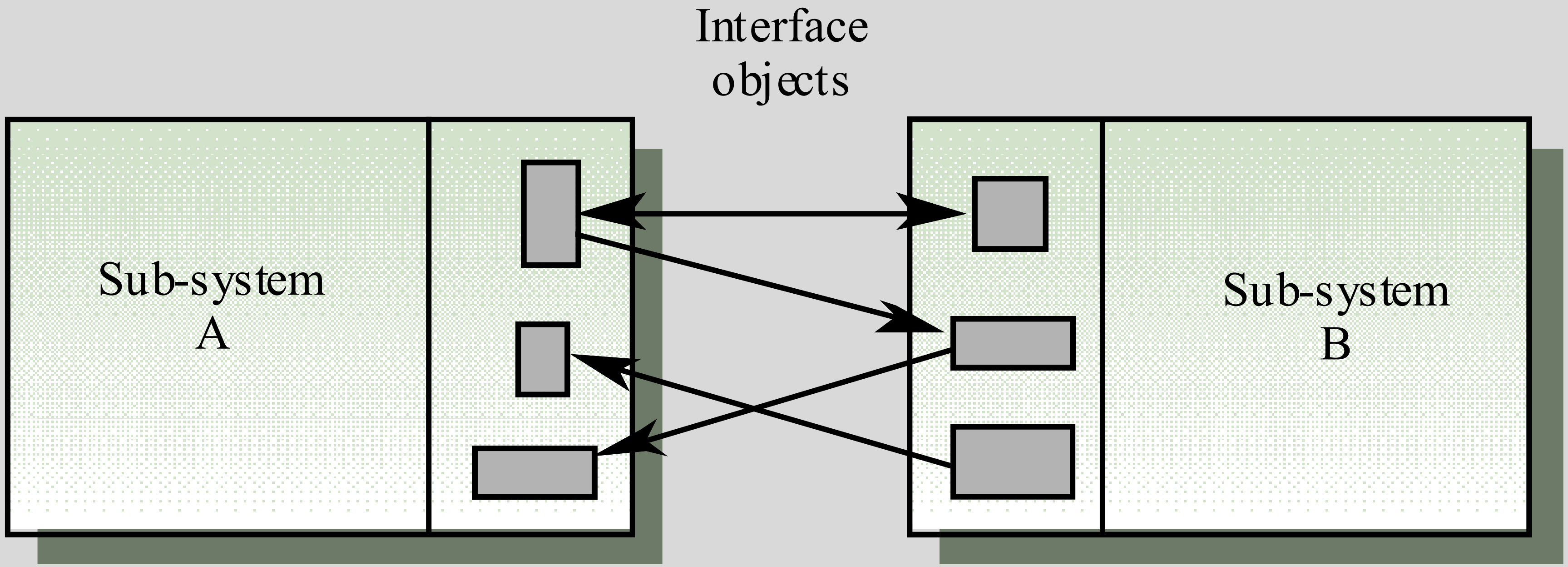
- Large systems are decomposed into subsystems with well-defined interfaces between these subsystems
- Specification of subsystem interfaces allows independent development of the different subsystems
- Interfaces may be defined as abstract data types or object classes

The algebraic approach to formal specification is particularly well-suited to interface specification

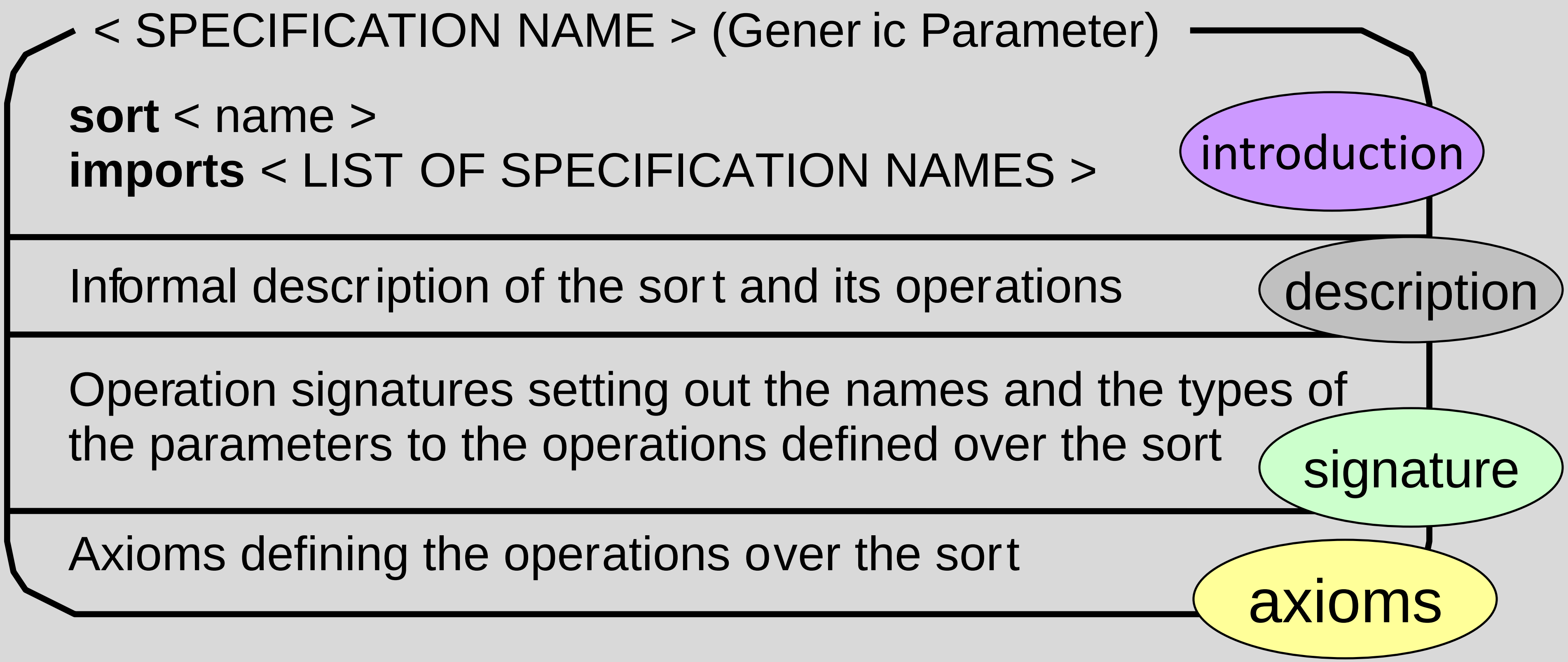
Sub-System Interface Specification

- Clear and unambiguous sub-system interface specifications reduce the chance of misunderstandings between a provider and user of a sub-system.
- The algebraic approach to specification was originally developed for the definition of *abstract data types*.
- This idea was then extended to model complete system specifications.

Sub-System Interfaces



The Structure of an Algebraic Specification



The Structure of an Algebraic Specification

- **Introduction** – Declares the sort (the type name) of the entity being specified, i.e., a set of objects with common characteristics. It also imports other specifications to use.
- **Description** – An informal description of the operations to aid understanding.
- **Signature** – Defines the syntax of the interface to the abstract data type (object), including their names, parameter list and return types.
- **Axioms** – Defines the semantics of the operations by defining axioms characterising the behaviour.

Systematic Algebraic Specification

- **Algebraic specifications** of a system may be developed in a systematic way:
  - Specification structuring;
  - Specification naming;
  - Operation selection;
  - Informal operation specification;
  - Syntax definition;
  - Axiom definition.

Specification Operations

- **Constructor operations**. Operations which create entities of the type being specified.
- **Inspection operations**. Operations which evaluate entities of the type being specified.

Example: Operations on a List ADT

- A list contains a sequence of elements of some type
- Elements may be added to the end and removed from the front
  - (this is also called a *queue*, how does this differ from a *stack*?).
- We want operations to :
  - Create
  - Cons (create a new list with an added member)
  - Head (to evaluate the first element)
  - Length
  - Tail (which creates a list by removing the first (head) element).

Example: Operations on a List ADT

- **Constructor operations** which evaluate to sort List
  - Create, Cons and Tail.
- **Inspection operations** which take sort list as a parameter and return some other sort
  - Head and Length.
- Tail can be defined using the simpler constructors Create and Cons.
- No need to define Head and Length with Tail.

Example: List Specification

List(Elem)
sort List
Imports INTEGER

Defines a list where elements are added at the end and removed from the front. The operations are Create, which brings an empty list into existence, Cons, which creates a new list with an added member, Length, which evaluates the list size, Head, which evaluates the list size, Head, which evaluates the front element of the list and Tail, which creates a list by removing the head from its input list.

Create -> List
Cons(List, Elem) -> List
Head (List) -> Elem
Length (List) -> Integer
Tail (List) -> List

Head(Create) = Undefined **exception** (empty List)
Head(Cons(L,v)) = **if** L = Create **then** v **else** Head (L)
Length(Create) = 0
Length(Cons(L,v)) = Length (L) + 1
Tail(Create) = Create
Tail(Cons(L,v)) = **if** L = Create **then** Create **else** Cons(Tail(L), v)

Lecture Key Points

- Formal system specification complements informal specification techniques.
- Formal specifications are precise and unambiguous. They remove areas of doubt in a specification.
- Formal specification forces an analysis of the system requirements at an early stage. Correcting errors at this stage is cheaper than modifying a delivered system.
- Formal specification techniques are most applicable in the development of critical systems and standards.