Advanced Software Engineering
(CS6401)

Autumn Semester (2024-2025)

Dr. Judhistir Mahapatro
Department of Computer Science and Engineering
National Institute of Technology Rourkela

Formal Specification of Systems

Formal Specification

- Any failure of a safety-critical product might result in loss of life and property.

- How can we ensure that specification is trouble free?

  Answer: formal specification

Formal Specification Technique

- Formal specification techniques enable us to:
  - precisely specify a system
  - verify that a system is correctly implemented.

- We say a system is correctly implemented:
  - when it satisfies its specification.

What is a Formal Technique?

- A formal technique is a mathematical method.

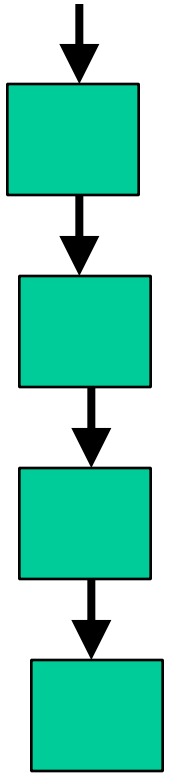
How is a formal technique useful?

- Formal techniques can be used:
  - to specify a hardware and/or software system.

- Formal techniques can also be used to:
  - verify whether a specification is realizable,
  - verify that an implementation satisfies its specification,
  - prove properties of a system without necessarily running the system, etc.

System development life-cycle

- The generally accepted paradigm for system development:
  - through a hierarchy of abstractions.



Formal Methods

- Each stage in this hierarchy:
  - An implementation of its preceding stage.
  - a specification of the succeeding stage.
- Formal techniques can be used:
  - at every stage of the system development activity.

Model versus Property-Oriented Methods

- Formal methods are usually classified into two broad categories:
  - model-oriented approach
  - property-oriented approach

Model-Oriented Style

- System specified by constructing its model:
  - in terms of mathematical structures:
    - tuples, relations, functions, sets, sequences, etc.
    - Also, state machines, Petri nets, etc.

Model-Oriented Style

- Well known model-oriented specification techniques are:
  - Z
  - CSP
  - VDM

Property-oriented style

- The system's behaviour is defined indirectly:
  - by stating its properties:
    - usually in the form of a set of axioms.
    - Examples: logic-based, algebraic specification, etc.

Example: producer/consumer system

- List the properties of the system:
  - the consumer can start consuming only after the producer has produced an item,
  - the producer starts to produce an item only after the consumer has consumed the last item.

Property-oriented specification

- Producing ==> No items exist for consumption
- consuming ==> Item exists for consumption

Example: producer/consumer system

- In a model-oriented approach:
  - Define the basic operations,
    - p(produce) and c(consume).
  - Then state that
    - S ==>S1+p,
    - S1 ==> S+c.

Comparison

- Property-oriented approaches are suitable for requirements specification:
  - model-oriented approaches are more suited to use in later phases of life cycle.
- Property-oriented approaches specify a system:
  - as a conjunction of axioms,
  - make it easier to alter/augment specifications at a later stage.

Comparison

- Model-oriented methods do not support logical conjunctions (and) and disjunctions (or):
  - even minor changes to a specification may lead to drastic changes to the entire specification.

Merits of Formal Methods

- Facilitates precise formulation of specifications
- Formal specifications encourage rigor.
  - the process of developing rigorous specification is often more important than specification.
- Construction of a rigorous specification clarifies several aspects of system behaviour:
  - which are not obvious in an informal specification.

Merits of Formal Methods

- It is actually cost-effective to spend more effort at the specification stage.
  - Otherwise many flaws that go unnoticed will appear at later stages of software development.

- For large and complex systems like real-time systems:
  - 80% of project costs and most of cost overruns result from iterative changes required due to improper requirements specification.

Merits of Formal Methods

- Additional effort required to construct a rigorous formal specification
- Formal methods usually have a well-founded mathematical basis.
  - formal specifications are precise
  - can be used to mathematically reason about the properties of a specification

Merits of Formal Methods

- Informal specifications are useful in understanding a system and its documentation:
  - But they cannot serve as a basis of verification.
  - Even carefully written informal specifications are prone to ambiguity and error.
- Formal methods have well-defined semantics.
  - Ambiguity in specifications is automatically avoided when one formally specifies a system.

Merits of Formal Methods

- Formal methods have mathematical basis:
  - scope for automating analysis of specifications.
  - Automatic verification of specifications
    - one of the most important advantages.
- Formal specifications can be executed:
  - provide immediate feedback on features of the specified system.
  - The concept of executable specification is related to rapid prototyping.

Limitations of Formal Methods

- Formal methods are difficult to learn and use.
- While using formal specifications
  - engineers tend to lose the overall perspective and get lost in the details.
- The basic incompleteness results of first-order logic (Gödel) suggest:
  - it is impossible to check absolute correctness of systems using theorem proving techniques.

Limitations of Formal Methods

- Formal techniques are not able to handle complex problems.
  - even moderately complicated problems blow up the complexity of formal specification and their analysis.
  - Also, a large unstructured set of mathematical formulas is difficult to understand.

Formal vs. Informal specification Methods

- Formal specifications:
  - do not replace informal descriptions.
  - but complement them.
- Comprehensibility of formal specifications is greatly enhanced when accompanied by an informal one.
- General recommendation:
  - Mixed approach
  - use formal techniques as a broad guideline for use of informal techniques.

Mixed Approach

- Formal method is used to identify verification steps:
  - but it is legitimate to apply informal reasoning in correctness arguments.
  - Any doubt or query relating to an informal argument is resolved by formal proofs.

Property-oriented Specifications

- Can be divided into two categories:
  - axiomatic specifications
  - algebraic specifications

Axiomatic Specifications

- Use first order logic:
  - write pre-conditions and post-conditions to specify operations

Axiomatic Specification of a Function

- Pre-conditions:
  - What are the requirements on the parameters of the function?
- Post-conditions:
  - What are the requirements when the function is completed?

Axiomatic Specification of a Function

- Domain:
  - What sort of things it acts upon?
- Co-domain (range):
  - What sort of answer does it give?

How to develop axiomatic specifications?

- Establish the range of input values over which the function should behave correctly:
  - establish input parameter constraints as a predicate.
- Specify a predicate defining the condition:
  - which must hold on the output of the function if it behaved properly.

How to develop axiomatic specifications?

- Establish what changes are made to the function's input parameters:
  - pure mathematical functions should not change its inputs, e.g., f(a,b,c)
  - programming languages allow function inputs to be modified by passing them as reference.
- Combine all of the above:
  - into pre and post conditions of the function

Axiomatic Specification: Example

- A function that takes a real number as argument and returns half the input value if the input is less than or equal to 100, or else returns double the value.

  - f(x : real) : real
  - Pre: $x \in R$
  - Post: $\{(x<=100) \wedge (f(x)=x/2)\} \vee \{(x>100) \wedge (f(x)=2*x)\}$

Example: Axiomatic Specification

- Consider a function search:
  - accepts parameters:
    - an array of integers
    - an integer key
  - returns array index of the number in array whose value equals key:
    - original input array is unchanged

Example: Axiomatic Specification

- search(X: intArray, key: integer): integer
- pre: there exists i ϵ [X_{first} …X_{last}], x[i]=key
- post: (X'[search(X, key)]=key) ^ (X=X')
- Error: search(X, key)=X'_{last}+ 1

Algebraic Specification

- An object class (or type) is specified in terms of:
  - Relationships existing between the operations defined on that type.
- It was first brought into prominence by Guttag:
  - Used for specification of abstract data types.
  - Various notations of algebraic specifications have evolved,
    - including OBJ and Larch languages.

Algebraic Specification

- Essentially algebraic specifications:
  - define a system as a heterogeneous algebra; i.e.,a collection of different sets on which several operations are defined.

Traditional Algebra

- Traditional algebra are homogeneous.
- A homogeneous algebra:
  - consists of a single set and several operations;
    e.g. { I, +, -, *, / }.

Algebraic Specification

- In contrast, consider:
  - alphabetic strings with operations:
    - concatenation and length,
  - not homogeneous algebra,
  - range of the length operation is the set of integers.

Algebraic Specification

- The collection of sets that form the heterogeneous algebra:
  - is called its signature.
  - For example consider {S, I, len, concat}
  - SXI is the signature.

Algebraic Specification

- To define a heterogeneous algebra, we need to specify:
  - its signature,
  - the involved operations, and their domains and ranges.

Algebraic Specification

- Algebraic specifications are usually presented in four parts:
  - types section
  - exceptions section
  - signature section
  - rewrite rules section

Types Section

- Types Section Lists:
  - sorts (or types) being specified
  - sorts being imported
  - Importing a sort:
    - makes it available in specification.

Exception Section

- Under normal conditions
  - the result of an operation may be of some sort.
- Under some exceptional conditions, the results may be something else.
- Lists names of exceptional conditions used in later sections:
  - under exceptional conditions error should be indicated.

Signature Section

- Defines signatures of interface procedures:
  - e.g. PUSH takes a stack and an element and returns a new stack.
  - push:
    - stack element stack

Rewrite rules section/ Equations Section

- Defines a set of rewrite rules:
  - Specifies what is always true about the behaviour of operations.

  - Lists the properties of the operators:
    - In the form of a set of axioms or rewrite rules.
      - allowed to have conditional expressions

Developing Algebraic Specification

- The first step in defining an algebraic specification:
  - identify the set of required operations.
  - e.g. for string, identify operations:
    - create, compare, concatenate, length, etc.

Developing Algebraic Specification

- Generally operations fall into 2 classes:
- Constructor Operations :
  - Operations which create or modify entities of the sort e.g., create, update, add, etc.
- Inspection Operations :
  - Operations which evaluate attributes of the sort, e.g., eval, get, etc.

Developing Algebraic Specification

- A rule of thumb for writing algebraic specifications:
  - first establish constructor and inspection operations
- Next, write down axioms:
  - compose each inspection operator over each constructor operator.

Developing Algebraic Specifications

- If there are *m* constructors and *n* inspection operators:
  - we should normally have *m***n* axioms.
  - However, an exception to this rule exists.

- If a constructor operation can be defined using other constructors:
  - we need to define inspection operations using only primitive constructors.

Another Perspective – Dangers of Physical Representations in Analysis

- Over Specification
  - Y2K problem, Postal code problem – tied in the logical to the physical representation

- Multiple representation
  - Stack as a array, list etc.
  - We may want to change the representation at a later point in time.

Towards Abstraction

- View an object by its operation, not by its representation
- Principle of selfishness:

If I am thirsty, an orange is something I can squeeze. If I am a painter, it is color which might inspire my palette, If I am a farmer, it is produce that I can sell at the market, but if I am none of these, and have no other use for the orange, then I should not talk about it, as the concept of orange does not for me even exist.

Mathematical Modeling

- Involves translating input space (I) to the output space (O) by a function F

  Ex: F(I) = O

- Program is a mathematical object and Language being used is a mathematical notation. Therefore, we can prove certain properties about it. For example, what the software do it is supposed to do and do it something that is not supposed to do.
- It will help us to prove this.

I/O Assertions

- Takes the form

S { P} Q

If S holds before the program P is executed then Q is going to hold after the program P executed.

Example:

sum = 0;

{ for i = 1 to n.

sum = sum + A_i

}

sum = $\sum_{i=1}^{n} A_i$

What does a Abstract Data Type (ADT) consist of ?

- Types
  - Collection of objects characterized by features, axioms
  - Specification is a declaration
  - Example: PrintingDevice

Definition of type of the object is going to be laid out. It can be a **parameterized type** (a **list** of **integers**) or **raw type** (example, a date).

What does a Abstract Data Type (ADT) consist of ?

- Operations/Functions
  - Operations applicable to the instances
  - Eg. Push: G, Stack(G) => Stack(G)
  - Multiple types
    - Creators (create new instances, example, date or stack)
    - Queries (used to extract the state whatever the data type already contains at a given point of time, read or manipulate the state)
    - Commands (setting the date)
    - Partial Functions
      - Cannot apply these under certain conditions
      - Example: Cannot pop on an empty stack, Cannot push on a full stack

Stack ADT

- Type

  Stack [G] - contains the elements of the type G

- Operations

  NEWSTACK -> STACK[G] returns a stack of type G

  PUSH: Stack[G], G -> STACK[G] -non declarative or non-imperative style of programming in ADT. Push operation takes two parameters (existing stack and an element) and it returns an other stack. Not manipulating the existing stack.

  POP: Stack[G] -> STACK[G]

  TOP: Stack[G] -> G

  isEmpty: Stack[G] -> BOOLEAN its query operation

PRECONDITION

POP : ISEMPTY --> FALSE //You cannot pop an empty stack

AXIOMS (Rules)

- ISEMPTY(NEWSTACK): -> TRUE
- POP(NEWSTACK) :-> ERROR
- TOP(NEWSTACK): -> ERROR
- TOP(PUSH(NEWSTACK, G)) -> G

Algebraic Manipulations using Axioms

- top(pop(push(pop(push(push(pop(push(push(push(new, x1), x2), x3)), top(pop(push(push(new, x4), x5)))), x6)), x7)))
- Work this one out using Axioms of the Stack ADT!

  - x4

- This is true no matter what representation of the stack that you choose to implement it.

Abstract Math to Concrete realizations

- ADTs to classes (module of OO design)
- Classes can be either of:
  - Deferred (partially implemented or abstract)
    - Useful in analysis and design
  - Effective (fully implemented or concrete)
    - A must in implementation

ADTs and Information Hiding

Public part:

ADT specification (E1)

Secret part:

- Choice of representation (E2)
- Implementation of functions by features(E3)

- Information hiding is an important aspect of software modularization. That part of the ADT or software module is needed for interaction with other module is needed to be exposed.
- Detailed Implementation of the functions need not to be exposed that to be differed.

Completeness of Specification

- Can we answer this question?
- Mathematically
  - If the axioms can be used to prove any other expression that can be constructed from the language of the theory
  - Well formed expressions only count
  - Push(G) is NOT well formed since it also needs a stack as an argument
  - Top (new) – is it well formed? If so, is it meaningful?
  - Structural integrity as well as semantic integrity needs to be checked for Query expressions and their importance in this evaluation

Example:

ReplaceITEM (Stack[G], G) -> Stack[G]

If (ISEMPTY = FALSE)

PUSH(POP(Stack[G]), G)

- ReplaceITEM is a compound operation whereas ISEMPTY, PUSH, POP are atomic operations.
- Syntactically meaningful expressions and semantically correct.

Definition: sufficient completeness

- An ADT specification for a type *T* is sufficiently complete if and only if the axioms of the theory make it possible to solve the following problems for any well-formed expression *e*:
  - S1: Determine whether *e* is correct.
  - S2: If *e* is a query expression and has been shown to be correct under S1, express *e*'s value under a form not involving any value of type T.

ADT Consistency

- An ADT specification is consistent if and only if, for any well-formed query expression *e*, the axioms make it possible to infer at most one value for *e*.

UNIX DIRECTORY

Type: UDIR

Operations:

/ : -> UDIR // root

mkdir : UDIR x NAME -> UDIR

// present working directory
and a name gives you
another new directory

cd : UDIR X ID -> UDIR

pwd: UDIR -> ID // ID : name or string (path)

isname: ID -> BOOLEAN // whether exist
within the directory

Axioms

PWD(/) = “/” ;

PWD(mkdir(D, i)) = PWD(D)

PWD (cd(D, i)) = PWD (i)

LIST (EXAMPLE2)

Type List[G]

Operations

CREATE -> List[G]

INSERT : List x i x G -> List[G] //list, index and item

DELETE : List x i -> List[G] //list and index

RETRIEVE: List x i -> G //list and index

ISEMPTY: List -> BOOLEAN

LENGTH: List -> INT // returns an integer

Axioms

ISEMPTY(CREATE) = TRUE

ISEMPTY(INSERT((CREATE), 1, G) = FALSE

Pros and Cons

- Algebraic specifications have a strong mathematical basis:
  - can be viewed as heterogeneous algebra.

- An important shortcoming of algebraic specifications:
  - cannot deal with side effects
  - difficult to use with common programming languages.

Pros and Cons

- Algebraic specifications are hard to understand:
  - also changing a single property of the system
    - may require changing several equations.

Summary

- We started by discussing some general concepts in formal specification techniques.
- Formal specifications have several positive characteristics.
  - the major shortcoming of formal techniques is that they are hard to use.

Summary

- It is possible that formal techniques will become more usable in future:
  - with the development of suitable front-ends.

- We discussed two sample specification techniques,
  - axiomatic specification
  - algebraic specification
  - give us a flavor of the issues involved in formal specification.