Formal Specification

Objectives

● To explain why formal specification techniques help discover problems in system requirements
● To describe the use of algebraic techniques for interface specification
● To describe the use of model-based techniques for behavioral specification
Formal methods

- Formal specification is part of a more general collection of techniques that are known as formal methods.
- These are all based on mathematical representation and analysis of software.
- Formal methods include
  - Formal specification;
  - Specification analysis and proof;
  - Transformational development;
  - Program verification.

Why formal methods?

REAL, EMPIRICAL WORLD  FORMAL, MATHEMATICAL WORLD

Empirical relational system → Measurement → Formal relational system

Implementation of solution

Empirical, relevant results

Mathematics, statistics

Interpretation

Numeric results
Formal Specifications

Every program can be represented by a flowchart:

![Flowchart Diagram]

Predicate true at this point in a program: S1 and S4 represent the specifications.

Basic mathematical model

Signatures: function: domain $\rightarrow$ range
- fun1: S1 and P1 $\rightarrow$ S3
- fun2: S1 and not(P1) $\rightarrow$ S3
- fun3: S3 and not(P3) $\rightarrow$ S4
- fun4: S3 and P3 $\rightarrow$ S4
- S2 and P2 == S4
- S2 and not(P2) == S3

A program is then some complex function C:

$$C(fun1, fun2, fun3, fun4, p1, p2, p3):S1 \rightarrow S4$$

If we could derive these relationships formally, then we have a mathematical proof that given S1 we would HAVE to end with S4.

Problem: Such proofs are hard.
Proof Models

Axiomatic verification (closest to previous figure) - Tony Hoare - 1969. Each program statement obeys a formal axiomatic definition. All models following this approach, more or less. This is the fundamental formal method.

Weakest precondition - Edsger Dijkstra - 1972. Similar to axiomatic verification, but allows for some non-determinism.


Others: Model checking, Petri nets, State machines, ...

Acceptance of formal methods

- Formal methods have not become mainstream software development techniques as was once predicted
  - A. Other software engineering techniques have been successful at increasing system quality. Hence the need for formal methods has been reduced;
  - B. Market changes have made time-to-market rather than software with a low error count the key factor. Formal methods do not reduce time to market;
  - C. The scope of formal methods is limited. They are not well-suited to specifying and analyzing user interfaces and user interaction;
  - D. Formal methods are still hard to scale up to large systems.

- One of these is false, another is unfortunately true, the other two are true. Which fit into each category?
Use of formal methods

- The principal benefits of formal methods are in reducing the number of faults in systems.
- Consequently, their main area of applicability is in critical systems engineering. There have been several successful projects where formal methods have been used in this area.
- In this area, the use of formal methods is most likely to be cost-effective because high system failure costs must be avoided.

Scalability of Formal Methods

Small grain methods (e.g., axiomatic, statement oriented) have a mathematical basis but are hard to scale up, and are hard to handle changes.

Large grain methods involve module composition allowing for simpler specifications, but are less precise. VDM and Z can be considered large grain methods.

Huge grain methods are typically systems. They may involve system-level protocols, such as TCP/IP, web browsers (HTML), ...
**Terminology**

**Validation** - Demonstrating that a program meets its specifications by testing the program against a series of data sets.

**Verification** - Demonstrating that a program meets its specifications by a formal argument based upon the structure of the program.

Validation assumes an execution of the program; verification generally does not.

---

**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 as made as the amount of rework due to requirements problems is reduced.
Cost profile

- The use of formal specification means that the cost profile of a project changes
  - There are greater up front costs as more time and effort are spent developing the specification;
  - However, implementation and validation costs should be reduced as the specification process reduces errors and ambiguities in the requirements.
  - We'll see this again with cleanroom

Example specification techniques

- Algebraic specification
  - The system is specified in terms of its operations and their relationships.
- Model-based specification
  - Specification is a mathematical proof of correctness
  - 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.
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 as it is focused on the defined operations in an object.

Specification operations

- **Constructor operations.** Operations which create entities of the type being specified.
- **Inspection operations.** Operations which evaluate entities of the type being specified.
- To specify behaviour, define the inspector operations for each constructor operation.
Algebraic data types

- Specification of a function, not implementation.
- Closely tied to abstract data type issues.

Example: stacks
- Functions: newstack, push, pop, top, empty, size

Axioms

1. pop(newstack) = newstack
2. pop(push(s,i)) = s
3. top(newstack) = undefined
4. top(push(s,i)) = i
5. empty(newstack) = true
6. empty(push(s,i)) = false
7. size(newstack) = 0
8. size(push(s,i)) = size(s)+1

newstack is a constructor, push is a generator, others operators.

Heuristic - combine each constructor and generator with each operator to get axioms.

We say that any implementation of the given functions that preserves these relationships is a valid implementation of a stack.
Proof methods - Data type induction

Similar to mathematical induction

- Let f be constructor, g generator, p predicate
- Show p(f) is true (base case)
- Show p(s) \rightarrow p(g(s))

Then true for all s.

Theorem: \text{size}(\text{push}(s,x)) > \text{size}(s)

P(s) is defined as \text{size}(\text{push}(s,x)) > \text{size}(s)

Data type induction proof

(Base case) s = newstack
Show: \text{size}(\text{push}(\text{newstack},i)) > \text{size}(\text{newstack})
\text{size}(\text{push}(\text{newstack},i)) = \text{size}(\text{newstack}) + 1 \quad \text{Axiom 8}
\text{size}(\text{newstack}) + 1 = 0 + 1 = 1 \quad \text{Axiom 7}
\text{size}(\text{push}(\text{newstack},i)) > \text{size}(\text{newstack})

Mathematical theorem, Axiom 7

(Generating case) s = \text{push}(s', x)
\text{size}(\text{push}(s,x)) > \text{size}(s) \quad \text{Goal}
\text{size}(\text{push}(\text{push}(s',x),x)) > \text{size}(\text{push}(s',x))

Definition of s
\text{size}(\text{push}(s',x)) + 1 > \text{size}(\text{push}(s',x)) \quad \text{Axiom 8}
1 + Y > Y \quad \text{Theorem}
Example 2 -- Addition

Axioms:
1. Add(0,x) = x
2. Add(succ(x),y) = succ(add(x,y))
0-consturctor, succ-generator, add-operator

Theorem: 1+1=2 (Note: 1=succ(0), 2=succ(1))

\[
\text{Add}(\text{succ}(0),\text{succ}(0)) = \text{succ}(\text{add}(0,\text{succ}(0))) \quad \text{Axiom 2}
\]
\[
\text{Add}(\text{succ}(0),\text{succ}(0)) = \text{succ}(\text{succ}(0)) \quad \text{Axiom 1}
\]
1+1=2  \quad \text{Substitution}

Example 3 - Associativity

\[
\text{add}(\text{add}(x,y),z) = \text{add}(x,\text{add}(y,z))
\]

Proof: Base case:
\[
\text{add}(\text{add}(0,y),z) = \text{add}(0,\text{add}(y,z)) \quad \text{Goal}
\]
\[
\text{add}(y,z) = \text{add}(y,z) \quad \text{Axiom 1}
\]

Inductive case: \[
\text{add}(\text{add}(x,y),z) = \text{add}(x,\text{add}(y,z))
\]
\[
\text{add}(\text{add}(\text{succ}(x),y),z) = \text{add}(\text{add}(\text{add}(x,y),z)) \quad \text{Axiom 2}
\]
\[
\text{add}(\text{add}(\text{succ}(x),y),z) = \text{succ}(\text{add}(\text{add}(x,y),z)) \quad \text{Axiom 2}
\]
\[
\text{add}(\text{add}(\text{succ}(x),y),z) = \text{succ}(\text{add}(x,\text{add}(y,z))) \quad \text{Ind. Hyp.}
\]
\[
\text{add}(\text{add}(\text{succ}(x),y),z) = \text{add}(\text{add}(x,\text{add}(y,z))) \quad \text{Axiom 2}
\]
AXIOMATIC VERIFICATION

{P1}S{P2} means that if P1 is true and S executes, then if S terminates, P2 will be true.

A
B Means if A is true then you can state that B is true.

Axioms of Logic

Composition

{P}S1{Q}, {Q}S2{R}
{P}S1; S2{R}

Consequence 1

{P}S{R}, R ⊃ Q
{P}S{Q}

Consequence 2

P ⊃ R, {R}S{Q}
{P}S{Q}

These embed programming constructs into mathematical logic.
### Statement axioms

**Conditional 1**
\[{P \& B}S(Q), P \& \neg(B) \Rightarrow Q \]
\{P\} if B then S\{Q\}

**Conditional 2**
\[{P \& B}S_1(Q), {P \& \neg(B)}S_2(Q) \]
\{P\} if B then S_1 else S_2\{Q\}

**While**
\[{P \& B}S(P)\]
\{P\} while B do S\{P \& \neg(B)\}
where P is an invariant

**Assignment**
\{P(e)x:=e(P(x))\} where P(x) is some predicate in x.

### Axiomatic Program Proof

Given program: s1; s2; s3; s4; ... sn and specifications P and Q:
- P is the precondition
- Q is the postcondition

Show that \{P\}s1;...sn\{Q\} by showing:
- \{P\}s1\{p1\}
- \{p1\}s2\{p2\}
- ... \{pn-1\}sn\{Q\}

Repeated applications of composition:
- \{P\}s1; ...; sn\{Q\}
Example: Prove the following

\{B \geq 0\}

1 \hspace{1em} X := B
2 \hspace{1em} Y := 0
3 \hspace{1em} while X > 0 do
4 \hspace{1em} \hspace{1em} begin
5 \hspace{1em} \hspace{1em} Y := Y + A
6 \hspace{1em} \hspace{1em} X := X - 1
7 \hspace{1em} \hspace{1em} end

\{Y = AB\}

Proof Outline

General method is to work backwards.
Look for loop invariant first:

- Y is part of partial product, and X is count of what remains
- So Y and XA should be the product needed, i.e., invariant
- \(Y + XA = AB\) is proposed invariant
- Try: \((Y + XA = AB \text{ and } X \geq 0)\)
Proof - Steps 1-4

\{(Y+(X-1)A=AB and X-1\geq 0)\} \ X:=X-1 \ \{(Y+XA=AB and X\geq 0)\} \\
Axiom of assignment (6)

\{((Y+A)+(X-1)A=AB and X-1\geq 0)\} \ Y:=Y+A \ \{(Y+(X-1)A=AB and X-1\geq 0)\} \\
Axiom of assignment (5)

\{(Y+A)+(X-1)A=AB and X-1\geq 0)\} \ Y:=Y+A; \ X:=X-1 \ \{(Y+XA=AB and X\geq 0)\} \\
Axiom of composition (5,6)

Y+XA=AB and X>0 \ \Rightarrow \ ((Y+A)+(X-1)A=AB and X-1\geq 0) \\
Mathematical Theorem

\{ Y+XA=AB and X>0\} \ Y:=Y+A; \ X:=X-1 \ \{(Y+XA=AB and X\geq 0)\} \\
Axiom of consequence
**Proof steps 5-8**

\((Y +XA = AB \text{ and } X \geq 0) \Rightarrow Y + XA = AB \text{ and } X > 0\)

*Mathematical theorem*

\{{{Y +XA = AB \text{ and } X \geq 0}) \land (X > 0)} Y := Y + A; X := X - 1 \{\{Y + XA = AB \text{ and } X \geq 0\}\}\}

**Axiom of consequence**

\{Y + XA = AB \text{ and } X \geq 0\} \text{ while } X > 0 \text{ do}

\(Y := Y + A; X := X - 1 \{Y + XA = AB \text{ and } X \geq 0\}\) \text{ and not}(X > 0) \quad \text{While axiom}

\((Y + XA = AB \text{ and } X \geq 0) \land \text{not}(X > 0)\) \Rightarrow (Y + XA = AB \text{ and } X = 0) \Rightarrow Y = AB \quad \text{Mathematical theorem}

---

**Proof - Steps 9-12**

\{Y + XA = AB \text{ and } X \geq 0\} \text{while } X > 0 \text{ do } Y := Y + A; X := X - 1 \{Y + AB\}

*Axiom of consequence*

\{0 + XA = AB \text{ and } X \geq 0\} Y := 0 \{Y + XA = AB \text{ and } X \geq 0\}

*Axiom of assignment*

\{0 + BA = AB \text{ and } B \geq 0\} X := B \{0 + XA = AB \text{ and } X \geq 0\}

*Axiom of assignment*

\{0 + BA = AB \text{ and } B \geq 0\} X := B; Y := 0 \{Y + XA = AB \text{ and } X \geq 0\}

*Axiom of composition*
Proof - Steps 13-15

\[(B \geq 0) \Rightarrow 0 + BA = AB \text{ and } B \geq 0\]

\{ B \geq 0 \} X := B; Y := 0 \{ Y + XA = AB \text{ and } X \geq 0 \}\}

\{ B \geq 0 \} \text{ entire program} \{ Y = AB \}

Summary - Axiomatic verification

Need to develop axioms for all language features. Can do it for simple arrays, procedure calls, parameters
  • Difficult, even on small programs
  • Very hard to scale up to larger programs
  • Does not translate well to non-mathematical problems
  • Extremely expensive to implement
  • Hard to automate. Manual process leads to many errors.
  • A cult following - “Every program must formally verified”
**BUT**

- Precise - clearly delineates the “what” with the “how”

- Basis for most other verification methods, including semiformal specification notations.

- For critical applications, it may be worth the cost.

---

**Behavioral specification**

- Algebraic specification can be cumbersome when the object operations are not independent of the object state.

- Model-based specification exposes the system state and defines the operations in terms of changes to that state.

- The Z notation is a mature technique for model-based specification. It combines formal and informal description and uses graphical highlighting when presenting specifications.
Specification language Z

Defines the data, types, functions, and relationships among these in a specification

General syntax:

<table>
<thead>
<tr>
<th>Component name</th>
<th>State information</th>
</tr>
</thead>
<tbody>
<tr>
<td>data1: type</td>
<td>data2: type</td>
</tr>
</tbody>
</table>

functional relationships among data objects:

Predicates

Some notation

Ξ - State X doesn't change
Δ - State X changes
ΘS = ΘS - S invariant
PX - a set of X; FX - a finite set of X
dom - domain; ran - range
f: X → Y - a function from X to Y
f: X ↦ Y - a partial injection (Not every X is defined and each Y has a unique X)
f ⊕ {X ↦ Y} - f appended with f(X) = Y
{X} < f - f with x removed from domain of f
x' - final value of x
x? - x is an input data item
x! - x is an output data item
Z Example - MSWE 607 grading

ClassRegistration

- ClassRoster
  - EnrolledInUMUC: F students
  - EnrolledInUMCP: F students
  - UMUCClasses: F ClassNumbers
  - UMCPClasses: F ClassNumbers
  - InUMUC: EnrolledInUMUC \rightarrow UMUCClasses
  - InUMCP: EnrolledInUMCP \rightarrow UMCPClasses
  - ClassList!: F students
  - UMCPClassList: F students

ClassList = \{x: EnrolledInUMUC \mid \text{InUMUC}(x) \cap \{607\} \neq \emptyset\}
UMCPClassList = \{x: EnrolledInUMCP \mid \text{InUMCP}(x) \cap \{607\} \neq \emptyset\}
UMCPClassList = ClassList!

Z specification - 2

PaperGradePart

- ClassRoster
  - ClassList: F students
  - PaperGrade?: ClassList \rightarrow N
  - PartialGrade!: ClassList \rightarrow \{\text{Letter} \mid \rightarrow N\}

\text{dom}(\text{PartialGrade}) = \text{ClassList}
\{x: \text{ClassList} \mid \text{PartialGrade}(x) = \{p: \rightarrow \text{PaperGrade}(x)\}\}
**Z specification - 3**

Δ ClassRoster  
ClassList: F students  
ExamOneGrade?: ClassList →→ N  
PartialGrade!: ClassList →→ F [Letter |→→ N]

\{x: ClassList | PartialGrade!(x) = PartialGrade(x) \oplus \{e \mid \rightarrow ExamOneGrade?(x)\}\}
Z specification - 5

Δ ClassRoster
ClassList: F students
ExamTwoGrade?: ClassList \rightarrow \mathbb{N}
PartialGrade!: ClassList \rightarrow F (Letter |\rightarrow \mathbb{N})

{x: ClassList | PartialGrade(x) = PartialGrade(x) \oplus (t \rightarrow \text{ExamTwoGrade?}(x))}

Z specification - 6

FinalGradePart

ClassRoster
ClassList: F students
Curve: R \rightarrow Letter
PartialGrade: ClassList \rightarrow F (Letter |\rightarrow \mathbb{N})
FinalGrade!: ClassList \rightarrow Letter

dom(FinalGrade!) = ClassList
ran(FinalGrade!) = \{A,B,C,F\}

\{x: ClassList | dom(PartialGrade(x)) = \{p,e,t,f\}
\text{Curve}(X) = (X \geq 90 \implies A) \cup (X \geq 80 \cap X < 90 \implies B) \cup (X \geq 70 \cap X < 80 \implies C) \cup (X < 70 \implies F)

\{x: ClassList | FinalGrade!(x) = \text{Curve}(0.10 \cdot \text{PartialGrade}(x)(p) + 0.25 \cdot \text{PartialGrade}(x)(e)
+ 0.25 \cdot \text{PartialGrade}(x)(t) + 0.40 \cdot \text{PartialGrade}(x)(f))}
Z summary

- Precise description of functionality
- Have we proven the functionality correct?
  - We’ve replaced low level code with more abstract descriptions of the functionality
  - Need to verify these, but easier to do than the previous axiomatic example

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.
Key points

- Algebraic techniques are suited to interface specification where the interface is defined as a set of object classes.
- Model-based techniques model the system using sets and functions. This simplifies some types of behavioural specification.
- Operations are defined in a model-based spec. by defining pre and post conditions on the system state.