Introducing Formal Methods

Formal Methods for Software
Specification and Analysis:
An Overview

L 5

Software Engineering and Formal Methods

- Every Software engineering methodology is based on a recommended development process proceeding through several phases:
  - » Analysis,Specification,Design,Coding,Unit Testing, Integration and System Testing, Maintenance
- Formal methods can:
  - » Be a foundation for describing complex systems
  - » Be a foundation for reasoning about systems
  - » Provide support for program development
- Complimentary approach to methodology!

Testing: Static vs Dynamic Analysis

- Static analysis of code‡Does not require execution of code
  - » Lexical analysis of the program syntax and investigates and checks the structure and usage of individual statements; often automated
- Dynamic Analysis of code‡Involves running the system (testing)
  - » Program run formally under controlled conditions with specific results expected
  - » Path and Branch Testing

What are Formal Methods?

- Techniques and tools based on mathematics and formal logic
- Can assume various forms and levels of rigor

least rigorous spectrum of rigor **most rigorous**

Occasional mathematical notation embedded in English specifications

languages with a precise semantics

Fully formal specification

Why Consider Formal Methods?

- Systems are increasingly dependent on software

System
Complexity

in
software-intensive
systems is very difficult

Time

L 5

components

- Complexity of systems with embedded software has increased rapidly
- Maintaining reliability

5

Formal Methods Concepts

Formal Specification Methods

Formal Formal Model Abstraction

Formal Specifications

- Translation of a non-mathematical description (diagrams, tables, English text) into a formal specification language
- Concise description of high-level behavior and properties of a system
- Well-defined language semantics support formal deduction about specification

Types of Specifications I

- Informal
  - » Free form, natural language
  - » Ambiguity and lack of organization can lead to incompleteness, inconsistency, and misunderstandings
- Formatted
  - » Standardized Syntax

» Basic consistency and completeness checks

» Imprecise semantics implies other sources of error may still be present

Types of Specifications II

■ Formal

» Syntax and semantics rigorously defined » Precise form, perhaps mathematical

» Eliminate imprecision and ambiguity

» Provide basis for mathematically verifying equivalence between specification and implementation

» May be hard to read without training

» Semantic distance?

Formal Specifications

■ Goal: Describe external behaviour without describing or constraining implementation ■ Formal Method has 2 parts:

» Logical Theory: Means by which one reasons about specifications, properties and programs

– First order predicate calculus (quantification over variables) – Second order predicate calculus (quantification over relations) – Temporal logic

» Structuring Theory: Defines elements being reasoned about

Types of Formal Specifications

- Property Oriented: State desired properties in a purely declarative way
  - » Algebraic: Data type viewed as an algebra, axioms state properties of data type's operations
  - » Axiomatic: Uses first order predicate logic, pre and post conditions Operational Specification: Describe desired behaviour by providing model of system
- Model Oriented: Provide direct way of describing system behaviour (sets, sequences, tuples, maps) :
  - » Abstract Model (in terms previously defined mathematical objects eg. sets, sequences, functions, mappings)
  - » State machines

Property Oriented: Algebraic Specifications

- Uses
  - » Input-Output Assertions

» Sets of operations

» Axioms specifying behaviour of

operations ■ Two parts to a specification

» syntax

» axioms

Model Oriented: Abstract Model Specifications

- Build an abstract model of required software behaviour using mathematically defined types (sets, relations)
- Define operations by showing effects of that operation on the model
- Specification includes:
  - » Model Type
  - » Invariant properties of model
  - » For each operation
    - – Name, parameters, return values
  - » Pre- and Post- conditions

Example Problem:
The English Specification

A space platform contains a number of instruments. Several communications channels are provided allowing both input and output instrument communications. Platform instruments may be placed in active or inactive states. Only active instruments may be assigned to I/O-channels. Active instruments may be assigned to more than one I/O-channel, up to some maximum number of I/O-channels per instrument. Further, I/O-channels may be shared by several active instruments, up to some maximum number of instruments shared per I/O-channel.

The objective is to construct a specification, that will manage the assignment of communication I/O-channels to spacecraft platform instruments.

L 5

14

Example Assignments

A B

C

D

Instruments

C1

C2

C3

I/O Channels

L 5

Basic Variables and Invariants

15

Z Specification:

■ Invariants are stated as the predicate

IO_Channel_Assignments

Basic_Types
active_instruments : *P* Platform_Instruments
assigned_to :
Communications_Channels « Platform_Instruments
available, busy: *P* Communications_Channels

range assigned_to [subset of] active_instruments
available Ç busy = f

Completed Make_An_Assignment$_0$ Schema

Make_An_Assignment$_0$
Δ IO_Channel_Assignments
instrument? : Platform_Instruments
channel? : Communications_Channels

instrument? ∈ active_instruments
channel? available
#(assigned_to > {instrument?}) < Max_Channels
channel? -> instrument? assigned_to

active_instruments' = active_instruments
assigned_to' = assigned_to {channel?->instrument?}

[#({channel?} < assigned_to) < Max_Instruments-1] \/
[#({channel?} < assigned_to) = Max_Instruments-1
/\ available' = available - {channel?}

/\ busy' = busy {channel?}]

Z Specification:
Schema for Error Condition

Instrument_Not_Active

Χ IO_Channel_Assignments
instrument? : Platform_Instruments
message! : Possible_Message

instrument? Ï active_instruments
message! = instrument_not_active

Interesting Thought Problem

- Alice and Bill throw a party, and invite 4 other couples. As each couple arrived, there are greetings and handshakes. At the end of the party, Bill asked everyone, including Alice, how many people they shook hands with: Every answer was different!
- How many hands did Alice shake?
  - »N.B: No one shakes hands with their own

partner or themselves. You greet a person only once.

Formal Methods Concepts

Formal Specification Methods

Formal Formal Model Abstraction
Specifications Proofs Checking

Formal Proofs

■ Complete and convincing argument for validity of some property of the system description ■ Constructed as a series of steps, each of which is justified from a small set of rules

■ Eliminates ambiguity and subjectivity inherent when drawing informal conclusions

■ May be manual but usually constructed with automated assistance

Model Checking

- Operational rather than analytic
- State machine model of a system is expressed in a suitable language
- Model checker determines if the given finite state machine model satisfies requirements expressed as formulas in a given logic
- Basic method is to explore all reachable paths in a computational tree derived from the state machine model

Abstraction

- Simplify and ignore irrelevant details
- Focus on and generalize important central properties and characteristics
- Avoid premature commitment to design and implementation choices

Program as Mathematical Object

■ Program Mathematical Object ■ Programming Language Mathematical Language

■ Can prove properties about the program

$I \overset{F}{\rightarrow} O$ F(I)‡O

Formal Specification Languages

■ Based on formal mathematical logic, with some programming language enhancements (such as type systems and parameterization)

■ Generally non-executable -- designed to specify what is to be computed, not how the computation is to accomplished

■ Most are based on axiomatic set theory or higher-order logic

Features of Specification Languages

- Explicit semantics. Language must have a mathematically secure basis.
- Expressiveness
  - » flexibility
  - » convenience
  - » economy of expression
- Programming language data types
  - » records
  - » tuples
  - » etc.

Features of Specification Languages (cont’d)

- Convenient syntax
- Diagrammatic notation
- Strong typing
  - » can be much richer than programming languages
  - » provides economy and clarity of expression
  - » type-checking provides consistency checks

- Total vs. partial functions
  - » most logics assume total functions
  - » subtypes can help make total functions more flexible

Features of Specification Languages (cont'd)

- Axioms and definitions
  - » axioms can introduce inconsistencies and should be used judiciously
  - » definitional principle assures that definitions are well-formed
  - » in some languages (e.g. PVS) type-checking conditions (to be proved) will be generated to assure sound definitions
- Modularization
  - » breaking a specification into modules is an important organizational feature
  - » parameterized modules allow reusability

Features of Specification Languages (cont'd)

- Built-in model of computation
  - » to discharge simple type-checking constraints » enhance proof-checking
- Maturity
  - » documentation

» tool support
» associated literature
» libraries of proven specifications
» some measure of standardization

Logical Errors in Formal Specifications

Logical inconsistency. Easiest logical errors to detect.

specification identify all contingencies and specify appropriate behavior for all cases? Peer review can aid in detection.

Accuracy -- does the specification mean what is intended? System invariants can help in detection.

Completeness -- does the

Techniques for Detection of Errors in Formal Specifications

The following error detection techniques are listed in increasing order of rigor and cost of application.
■ Inspection of the formal specification (manual) ■ Parsing for syntactic correctness (automated) ■ Type-checking for semantic consistency (automated) ■ Simulation/animation based on the specification (automated). Only possible if the language provides an execution option.

■ Theorem proving, proof-checking, model-checking for logical anomalies

Formal Specifications as a System Description

■ Clarify requirements and high-level design

■ Articulate implicit assumptions

■ Identify undocumented or unexpected assumptions

■ Expose flaws

■ Identify exceptions

■ Evaluate test coverage

Benefits of Formal Specifications

- Higher level of rigor enables a better understanding of the problem
  - Defects are uncovered that would likely go unnoticed with traditional specification methods
- Identify defects earlier in life cycle
- Can guarantee the absence of certain defects

Benefits of Formal Specifications (cont'd)

- Formal specification language semantics allow checks for self-consistency of a problem specification
- Formal specifications enable formal proofs which can establish fundamental system properties and invariants
- Repeatable analysis means reasoning and conclusions can be checked by colleagues

Benefits of Formal Specifications (cont'd)

- Encourages an abstract view of system -- focusing on *what* a proposed system should accomplish as opposed to *how* to accomplish it
- Abstract formal view helps separate specification from design
- Enhances existing review processes by adding a degree of rigor

Limitations to Formal Methods

- Used as an adjunct to, not a replacement for, standard quality assurance methods
- Formal methods are not a panacea, but can increase confidence in a product's reliability if applied with care and skill
- Very useful for consistency checks, but can not assure completeness of a specification

Cautions in the Use of Formal Methods

- Judicious application to suitable project environments is critical if benefits are to exceed costs
- FM and problem domain expertise must be fully integrated to achieve positive results

Conclusion

- FM are no panacea
- FM can detect defects earlier in life cycle
- FM can be applied at various levels of resource investment
- FM can be integrated within existing project process models
- FM can improve quality assurance when applied judiciously to appropriate projects