

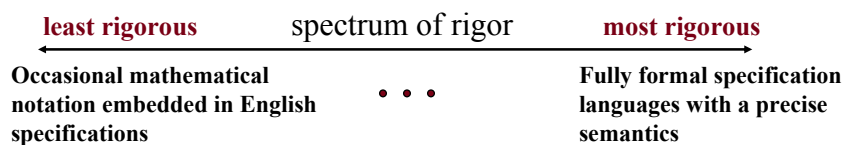
Introducing Formal Methods

Formal Methods for Software
Specification and Analysis:
An Overview

1

What are Formal Methods?

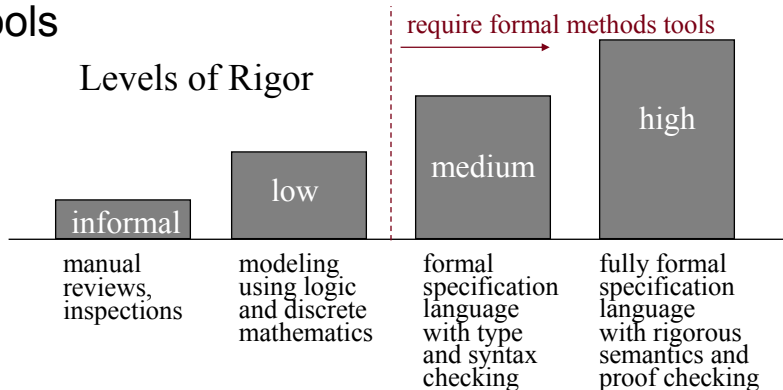
- Techniques and tools based on mathematics and formal logic
- Can assume various forms and levels of rigor



2

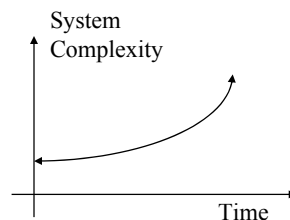
Level of Rigor for Formal Methods

Increasing rigor is usually associated with increasing dependence on automated support tools

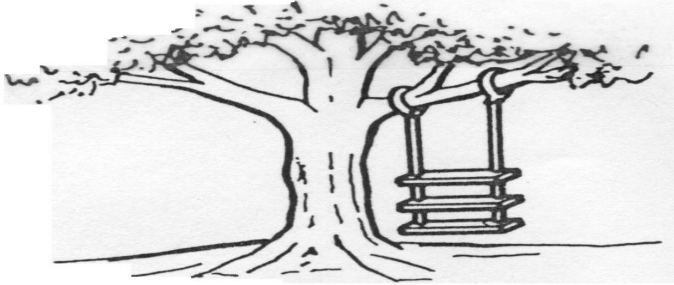


Why Consider Formal Methods?

- Systems are increasingly dependent on software components
- Complexity of systems with embedded software has increased rapidly
- Maintaining reliability in software-intensive systems is very difficult

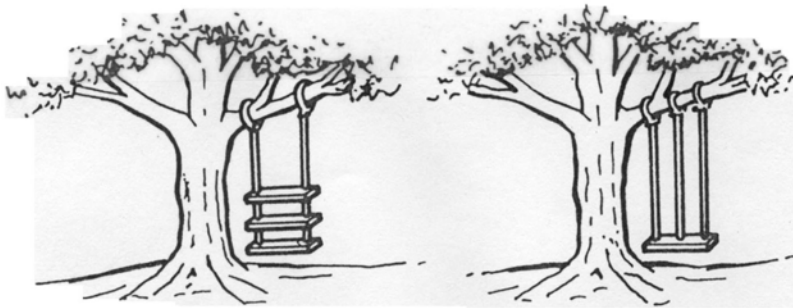


The Need for Formal Specification



**AS PROPOSED BY THE
PROJECT SPONSOR**

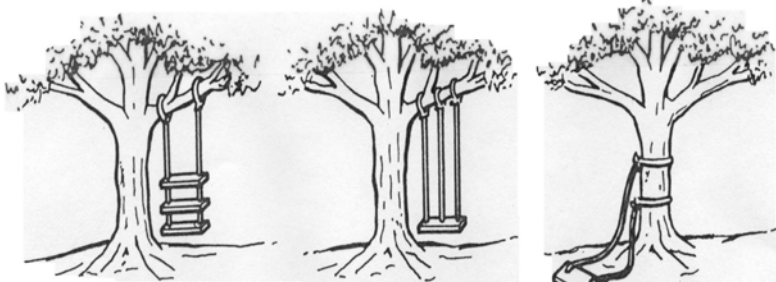
The Need for Formal Specification



**AS PROPOSED BY THE
PROJECT SPONSOR**

**AS SPECIFIED IN THE
PROJECT REQUEST**

The Need for Formal Specification



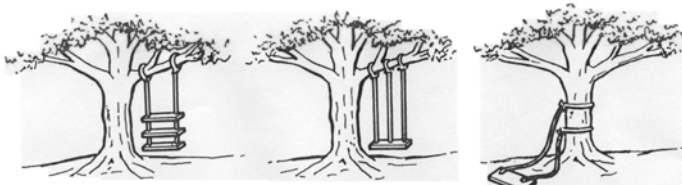
AS PROPOSED BY THE
PROJECT SPONSOR

AS SPECIFIED IN THE
PROJECT REQUEST

AS DESIGNED BY THE
SENIOR ANALYST

7

The Need for Formal Specification



AS PROPOSED BY THE
PROJECT SPONSOR

AS SPECIFIED IN THE
PROJECT REQUEST

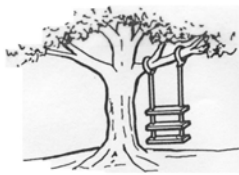
AS DESIGNED BY THE
SENIOR ANALYST



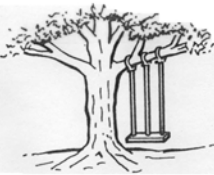
AS PRODUCED BY
THE PROGRAMMERS

8

The Need for Formal Specification



AS PROPOSED BY THE PROJECT SPONSOR



AS SPECIFIED IN THE PROJECT REQUEST



AS DESIGNED BY THE SENIOR ANALYST



AS PRODUCED BY THE PROGRAMMERS



AS INSTALLED AT THE USER'S SITE

The Need for Formal Specification



AS PROPOSED BY THE PROJECT SPONSOR



AS SPECIFIED IN THE PROJECT REQUEST



AS DESIGNED BY THE SENIOR ANALYST



AS PRODUCED BY THE PROGRAMMERS



AS INSTALLED AT THE USER'S SITE



WHAT THE USER WANTED

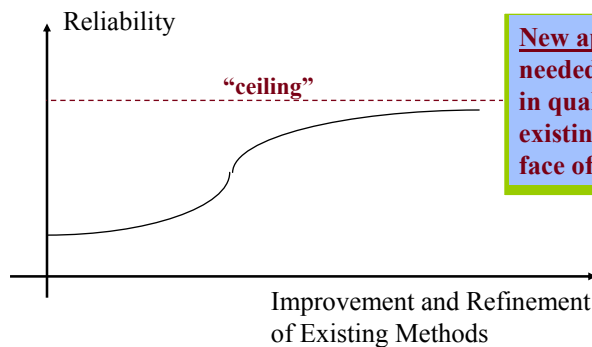
The Complexity Factor

- Fault protection and safety functions are no longer allocated solely to hardware
- Software must be able to detect and isolate failures, then execute recovery scenarios
- Software-intensive systems fail in ways characteristically different from hardware systems

11

Reliability and Traditional Methods

“Quality ceilings” are being encountered using traditional techniques



New approaches may be needed to achieve increases in quality (or to maintain existing quality levels in the face of increasing complexity)

12

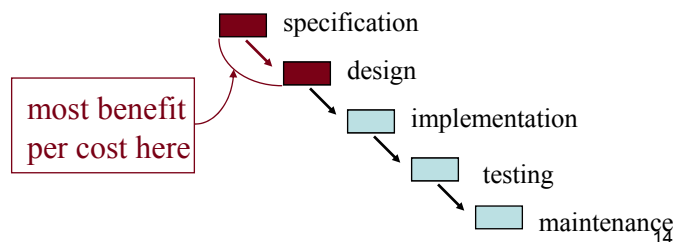
Safety Analysis

- Standard requirements analysis focuses on functional correctness
- Safety analysis focuses on identifying undesirable system behavior that would create an unsafe or hazardous condition
- Formal methods enhance the analyst's ability to model and check safety conditions

13

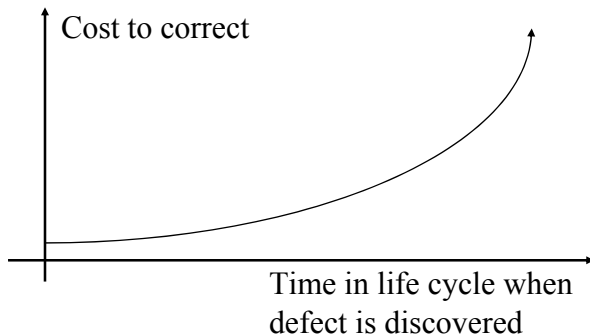
Choosing a Life Cycle Phase

- Formal methods can be applied to all phases of the life cycle
- However, benefit-to-cost ratio seems best in specification and high level design phases



Defects and Cost to Correct

Earlier detection of defects is essential for complex projects



15

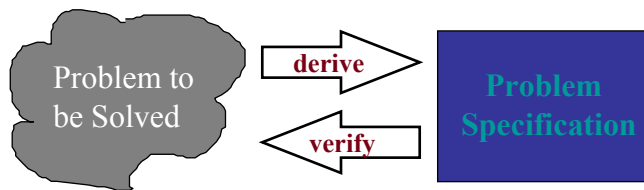
Formal Methods and the Specification Phase

- This phase is less automated and not tightly coupled to specific languages and notations
- Specification work products are less effectively analyzed than later phase products
- Thus, insertion of formal methods:
 - creates little intrusion on the existing process
 - can dramatically improve analysis capability

16

Difficulty in Verifying Specifications

Problem specification is difficult to verify because there is no preceding phase deliverable to be used as a benchmark



17

Informal Specifications Verification

- Informal verification analysis is subjective, often non-repeatable, and inherently unreliable
- Inductive nature of informal requirements verification runs risk that some scenarios and properties will not be covered or considered -- this risk is especially high for complex projects

18

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

19

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

20

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

21

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

22

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

23

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

24

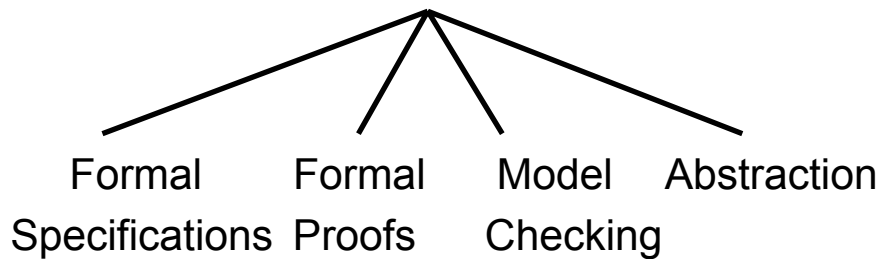
Introducing Formal Methods

Formal Methods Concepts

25

Formal Methods Concepts

Formal Specification Methods



26

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

27

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

28

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

29

Abstraction

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

30

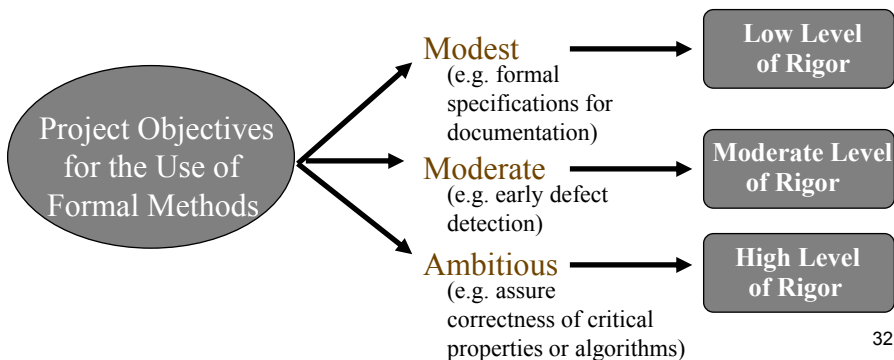
Establishing Formal Methods on a Project

- The use of formal methods is not an “all or nothing” proposition
- The level of rigor employed can be tailored to fit specific:
 - budgets,
 - schedules,
 - and technical environments
- Can be integrated into existing processes with some modifications

31

Types of Analysis/Formal Methods

The preferred type of analysis and method is strongly influenced by the project objectives

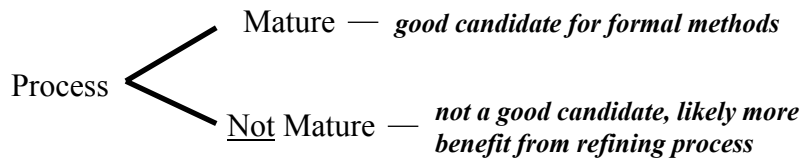


32

Process Prerequisites

Development process should be “mature,” specifically, the process should have:

- discrete phases
 - work products defined for each phase
 - analysis procedures in place for work products
 - scheduled review of work products
-



33

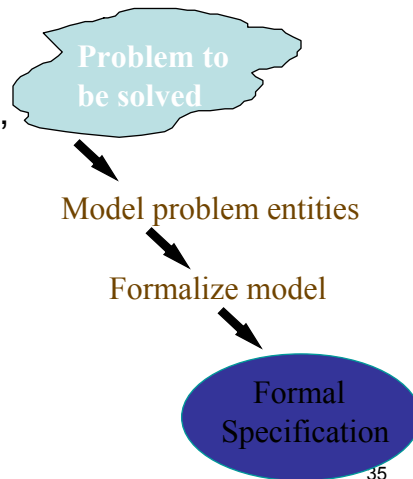
Process Integration Strategy

- If requirements analysis procedures are well-defined, few changes to the process will be required to integrate formal methods
- Formal methods can replace or complement steps within the existing process

34

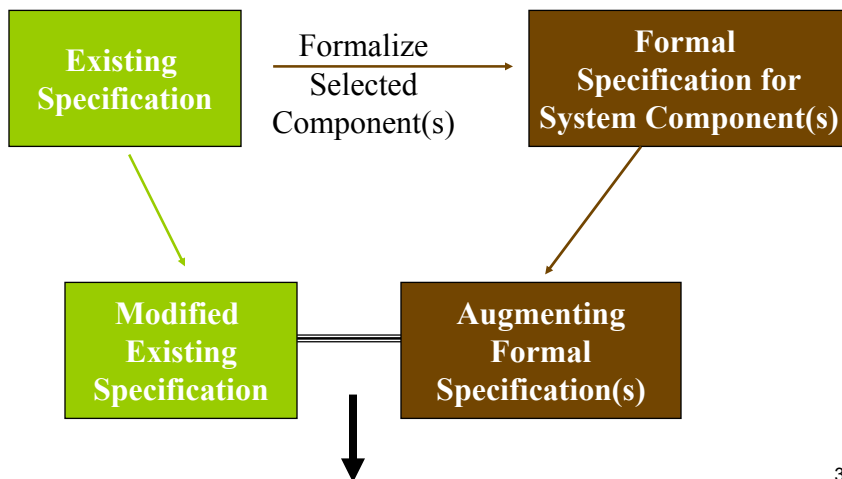
Process Modifications with Formal Specification Methods

- Initial modeling activity employs finite-state machines, object diagrams, etc.
- The model must be “formalized” or converted to a formal language
- Resulting work product is called a “formal specification”



35

Using Formal Methods with Existing Specifications



36

Summary

- Formal methods include a number of concepts/methods
 - Formal specifications
 - Formal proofs
 - Model checking
 - Abstraction
- Formal methods can be applied at various levels of rigor

37

Summary (cont'd)

- Formal methods should be employed in a mature, disciplined development environment
- Environment should exhibit a strong quality emphasis
- Adequate expertise, training, and support must be present
- Formal methods can be integrated into existing development processes

38

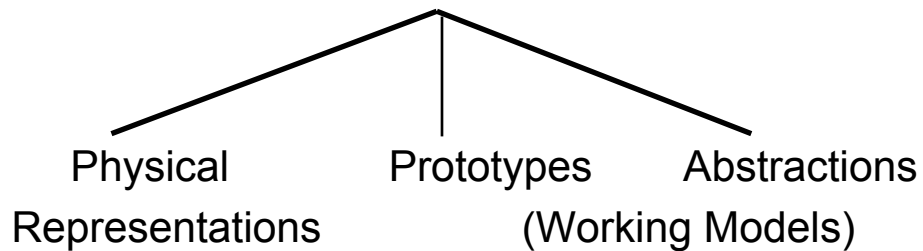
Introducing Formal Methods

Mathematical Models

39

Different Kinds of Models

Models of a System



40

Mathematical Models

- A mathematical model is an abstract representation of a system employing mathematical entities and concepts
- Model should be expressive enough to capture the essential characteristics of the system being modeled
- If the model is intended for deductive reasoning about the underlying system, it should provide sufficient analytic power for this purpose

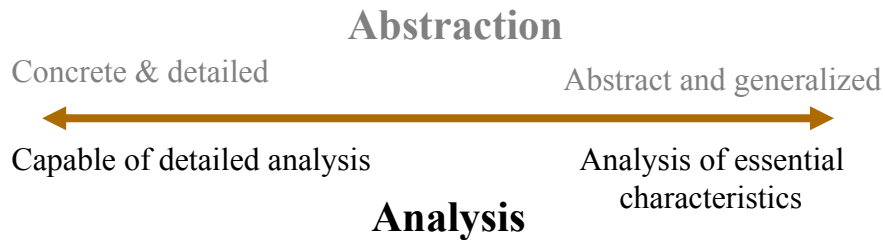
41

Characteristics of Mathematical Models

- Abstraction
- Focus/Scope
- Expressiveness vs. Analytic Power
- Intuitive vs. Non-intuitive Representation
- Accuracy

42

Abstraction vs. Analysis



43

Benefits of Mathematical Modeling

- Model is more concise and precise than natural language, pseudo-code, and diagrammatic representations
- Model can be used to calculate and predict system behavior
- Model can be analyzed using mathematical reasoning -- proving properties, deriving new behaviors, etc.

44

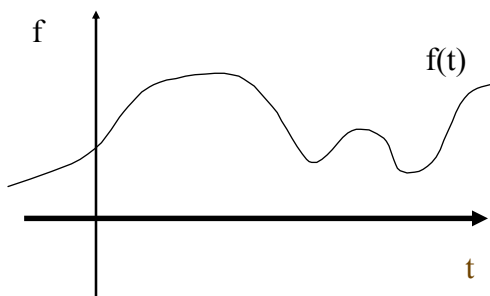
Two Major Types of Mathematical Models

- Continuous models
use the mathematics of continuous function theory -- derivatives, integrals, differential and integral equations
- Discrete models
use the mathematics of formal logic and set theory

45

Comparison of Continuous vs. Discrete Models

Continuous Modeling

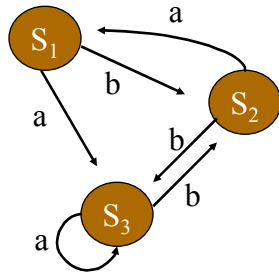


Function f describes the “state” of the modeled system at any given time t -- note there are an infinite number of possible states. Transition from one “state” to another is smooth (continuous) and computed using function f .

46

Comparison of Continuous vs. Discrete Models (cont'd)

Discrete Modeling



A transition function T governs movement from one of a finite number of system states (S_1, S_2, S_3 in this case) to another, depending upon the input (a or b in this case).

For example,

$$T(S_1, b) = S_2$$

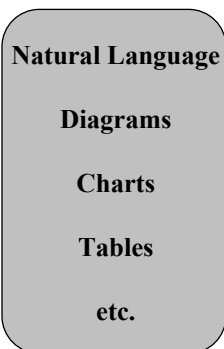
$$T(S_2, a) = S_1$$

$$T(S_3, a) = S_3$$

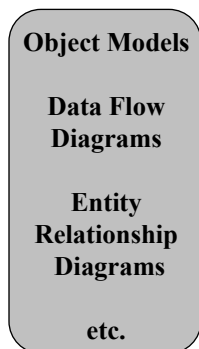
etc.

47

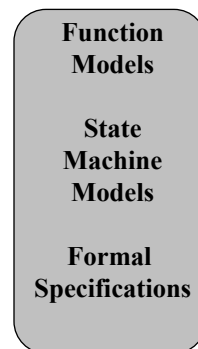
Discrete Modeling Techniques



Informal Models



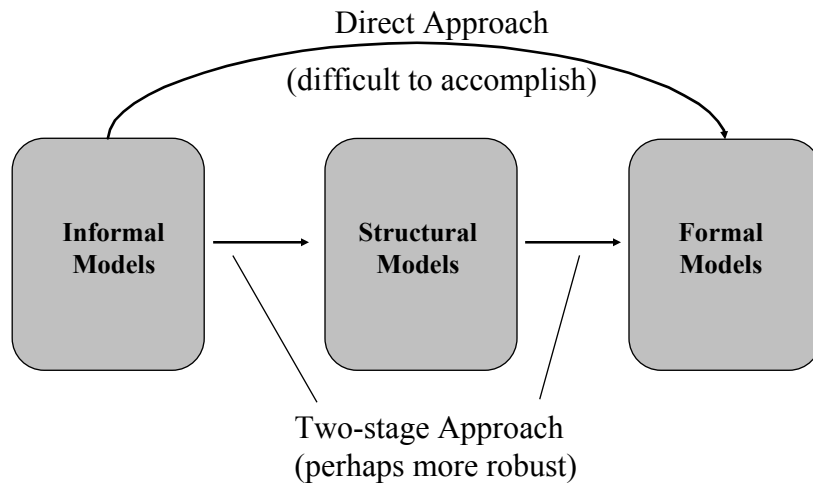
Structural Models
(Employ Formalisms)



Formal Models
(Employ Formal Semantics)

48

Discrete Modeling Techniques (cont'd)



49

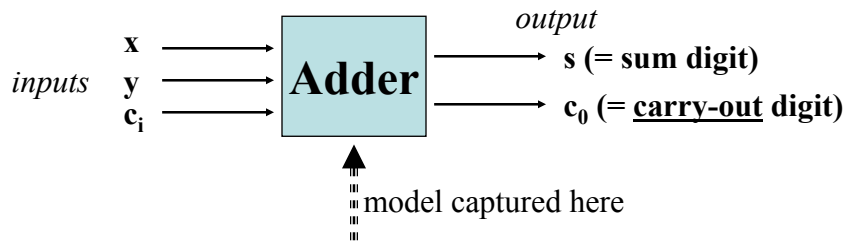
Object Models

- Object models represent systems as structured collections of classes and objects and relations between objects
- Object models provide a static view of a system
- Some object modeling methodologies employ additional modeling techniques such as entity-relationship diagrams, state machine diagrams, and dataflow diagrams, yielding a composite static/dynamic model
- Object models are structural in nature and are very useful for creating initial system models -- these may be mapped to a more formal model if desired

50

Function Model -- Example

A Full Binary Adder: Given input digits x and y and a previous carry bit (c_i for carry-in), compute the sum digit and new carry bit.

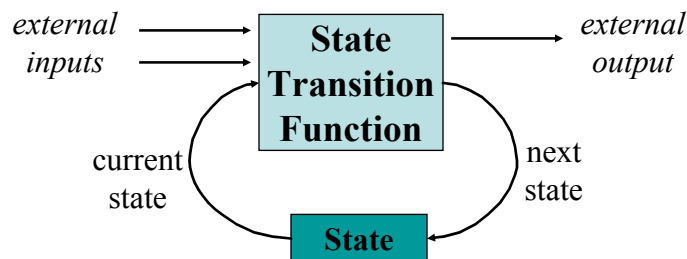


$$\text{Adder}(x, y, c_i) = (s, c_0) = ((x + y + c_i) \bmod 2, (x + y + c_i) \text{div } 2)$$

53

State Machine Models

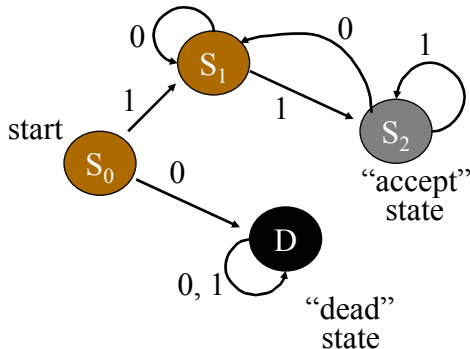
The essential characteristics of the system are modeled using the concept of "system state" and associated rules for transition between states



54

State Machine Model -- Example

A Simple String Parser: Given an input string of 0's and 1's, determine if the string starts and ends with a 1.



State Transition Function

		current state			
		S ₀	S ₁	S ₂	D
input	0	D	S ₁	S ₁	D
	1	S ₁	S ₂	S ₂	D
		next state			

55

Summary

- Abstract mathematical models can provide the ability to predict and check system behavior
- Discrete mathematical models are fundamental to formal methods
- Structural models (such as object models) provide a good intermediate step toward more formal models
- Discrete formal models are based on formal logic and allow mathematical reasoning about the modeled system
- Function and state machine discrete models are particularly useful in formal methods

56

Introducing Formal Methods

Formal Specification Languages
and Analysis

57

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

58

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.

59

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

60

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 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

61

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

62

Importance of Types

- Specification languages can have much richer type systems than programming languages, since types do not have to be implemented directly
- Type-checking can be used to detect faults and inconsistencies
- Essential features of a model can be embedded in types and subtypes

63

System Operations as Functions

- Basic system operations are often modeled as functions
- Functions can modify system state, hence invariance conditions are often imposed on appropriate combinations of functions (examples later)
- Theorems and axioms can be used to model other system invariants employing functions

64

Modeling with Functions -- Example

An abstract stack model (syntax is modified from PVS). The example is adapted from Rushby*.

We define a stack and the elements to be placed in the stack as unspecified types.

```
stack : TYPE
element : TYPE
```

We use a subtype to define a nonempty stack type.

```
nonempty_stack : TYPE = {s : stack | s /= empty}
```

Basic operations are now defined as functions.

```
push : [element, stack -> nonempty_stack]
pop : [nonempty_stack -> stack]
top : [nonempty_stack -> element]
```

* J. Rushby, *Formal Methods and the Certification of Critical Systems*, Tech. Report CSL-93-7, SRI, International, Menlo Park, CA

Modeling with Functions -- Example (cont'd)

The essential behavior of these three functions is stated in the following two axioms.

```
Pop_Push : AXIOM
  pop(push(e, s)) = s
```

```
Top_Push : AXIOM
  top(push(e, s)) = e
```

Type-checking now assures that the expression *pop(empty)* is never allowed. Likewise, the following theorem follows immediately from the type definitions.

```
Push_Empty : THEOREM
  push(e, s) /= empty
```

Modeling with Functions -- Example (cont'd)

Theorems can be used in PVS type-checking, adding a power capability to the type-checking system. The following discussion illustrates.

Consider whether the following equation (which we certainly wish to be true) will even pass type-checking.

$$\text{pop}(\text{pop}(\text{push}(x, \text{push}(y, s)))) = s \quad (1)$$

Assuming typical deterministic programming language style type-checking is used, we would conclude that there is a problem, because we cannot be assured using only the type definitions that the inner argument $\text{pop}(\text{push}(x, \text{push}(y, s)))$ in the expression on the left is not empty, as is required by the signature of the function pop . Thus the expression will not pass type-checking.

67

Modeling with Functions -- Example (cont'd)

However, using the axioms, we can prove the following theorem.

Theorem 1 : THEOREM
 $\text{pop}(\text{push}(x, \text{push}(y, s))) \neq \text{empty}$

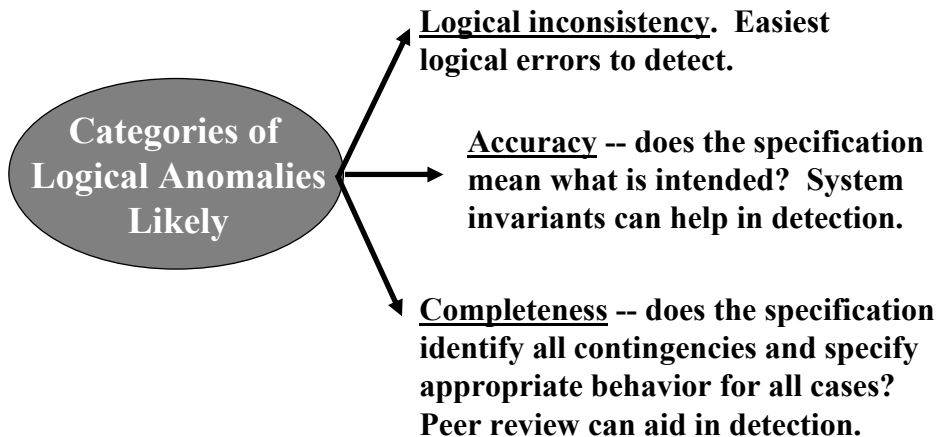
If the type-checking system can call on this theorem, then equation (1) can pass type-checking. Strong type-checking systems such as PVS can employ theorems in type-checking.

This is an important capability. Without it, we would have to modify the signature of our function pop to be able to prove equation (1). But the resulting weaker type definition would cause us to lose an opportunity for catching type inconsistencies and faults in our model.

The use of subtypes and theorems in type-checking puts at our disposal a very powerful mechanism for capturing requirements in type definitions.

68

Logical Errors in Formal Specifications



L.5

69

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

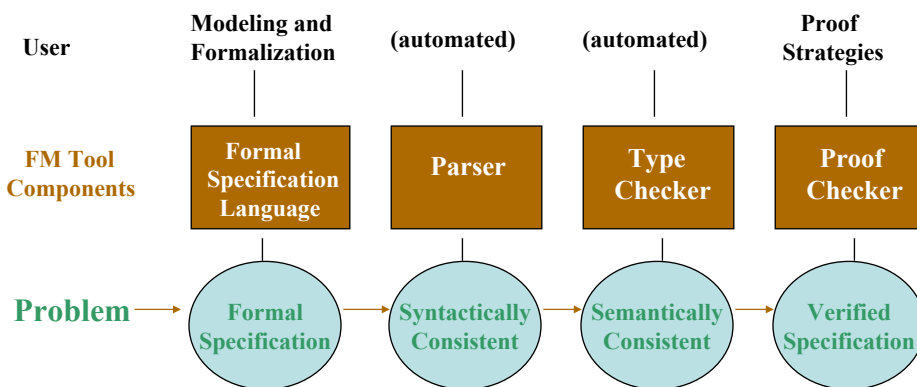
70

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

71

An Overview of Formal Methods Specification Tools



72

Summary

- Formal specification languages have explicit semantics and are based on formal mathematical logic. They also usually include a variety of programming language features for more expressiveness and convenience of use.
- A rich type system combined with functions can model many essential system characteristics
- Detection of errors is accomplished through a series of increasingly rigorous techniques
- Automated tools are employed for much error detection, augmented by manual inspection and review