

Formal Specification Methods

David S. Rosenblum ICS 221 Winter 2001



What Are Formal Methods?

- Use of formal notations ...
 - first-order logic, state machines, etc.
- ... in software system descriptions ...
 - system models, constraints, specifications, designs, etc.
- ... for a broad range of effects ...
 - correctness, reliability, safety, security, etc.
- ... and varying levels of use
- quidance, documentation, rigor, mechanisms

Formal method = specification language + formal reasoning



Objectives of Formal Methods

- Verification
 - "Are we building the system right?"
 - Formal consistency between specificand (the thing being specified) and specification
- Validation
 - "Are we building the right system?"
 - Testing for satisfaction of ultimate customer intent
- Documentation
 - Communication among stakeholders



Why Use Formal Methods?

- Formal methods have the potential to improve both software quality and development productivity
 - Circumvent problems in traditional practices
 - Promote insight and understanding
 - Enhance early error detection
 - Develop safe, reliable, secure software-intensive systems
 - Facilitate verifiability of implementation
 - Enable powerful analyses
 - simulation, animation, proof, execution, transformation
 - Gain competitive advantage



Why Choose Not to Use Formal Methods?

- Emerging technology with unclear payoff
- Lack of experience and evidence of success
- Lack of automated support
- Lack of user friendly tools
- Ignorance of advances
- High learning curve
- Requires perfection and mathematical sophistication
- Techniques not widely applicable
- Techniques not scalable
- Too many in-place tools and techniques



Desirable Properties of Formal Specifications

- Unambiguous
 - Exactly one specificand (set) satisfies it
- Consistency
 - A specificand exists that satisfies it
- Completeness
 - All aspects of specificands are specified
- Inference
 - Consequences of the specification and properties of its satisfying specificands are discovered



Different Kinds of Formal Specification Languages

- Axiomatic specifications
 - Defines system in terms of logical assertions
- State transition specifications
 - Defines system in terms of states & transitions
- Abstract model specifications
 - Defines system in terms of mathematical model
- Algebraic specifications
 - Defines system in terms of equivalence relations
- Temporal logic specifications
- Defines operations in terms of time-ordered assertions
- Concurrency specifications
 - Defines operations in terms of concurrent event occurrences



Tool Support for Specification Languages

- Modeling
 - Editors and word processors
- **Analysis**
 - Syntax checking
 - Model checking
 - Proving and proof checking

 - Property checking
 deadlock, reachability, data flow, liveness, safety, ...
- Runtime checking
- Synthesis
 - Refinement
 - Code generation
 - Test case and test oracle generation



A Closer Look: Axiomatic Specification

- Formal specification in which statements in first-order predicate logic are used to define the semantics of a system and its constituent elements (statements, functions, modules)
- Usually taken to mean specification with
 - Pre-conditions
 - Post-conditions
 - Invariants
 - Point Assertions



History of Axiomatic Specification

- Attempts to put program development on a formal basis date at least to John McCarthy's 1962 paper (w.r.t recursive functions)
- Floyd's 1967 paper presented the first worked-out approach (in terms of flowcharts)
- Hoare's 1969 paper formed the basis for much of the later work in formalized development
 - Formal specification languages
 - Formal verification
 - Axiomatic semantics of programming languages



Hoare's Basic Approach

P {S} Q (nowadays written {P} S {Q})

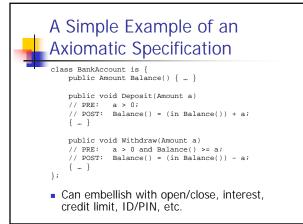
- If environment of S makes assertion P true
- And if S terminates
- Then assertion Q must be true
- - If the environment doesn't establish P, Q need not be true
 - If S doesn't terminate, Q need not be true
 - Proving {P} S {Q} establishes partial correctness
 - To establish total correctness, one must also prove that S terminates, which in general is undecidable

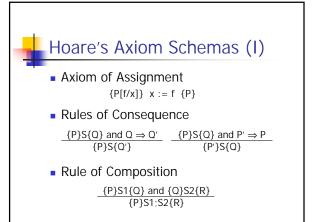


Axiomatic Specification of **Programs**

{P} S {Q}

- One typically *specifies* (components of) whole
 - S is a program, module, method, etc.
 - P is the desired pre-condition of S
 - Q is the desired post-condition of S
- The axiomatic semantics of the language of S comprises Hoare-style axiom schemas for the constituent statements of S
 - assignments, conditionals, loops, etc.
 - Used for verifying S with respect to P and Q

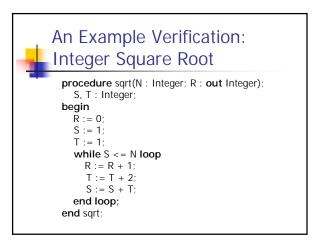






Hoare's Axiom Schemas (II)

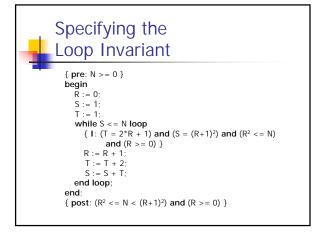
- Rule of Iteration
 - P and {C}S{P}
 - {P}while C do S{not C and P}
 P is the *loop invariant*, which typically must
 - be supplied by the specifier
- Rules have been defined for other common language features
 - arrays, do-until, if-then, if-then-else, subprogram calls, ...



```
Specifying the

Pre- and Post-Conditions

{ pre: N >= 0 }
    begin
    R := 0;
    S := 1;
    T := 1;
    while S <= N loop
    R := R + 1;
    T := T + 2;
    S := S + T;
    end loop;
end;
    { post: (R<sup>2</sup> <= N < (R+1)<sup>2</sup>) and (R >= 0) }
```

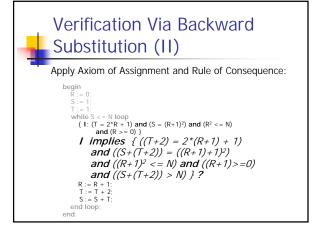




Verification Via Backward Substitution (I)

Apply Rule of Iteration and Rule of Consequence:

```
end loop;
 \{(T = 2*R + 1) \text{ and } (S = (R+1)^2) \text{ and } (R^2 <= N)\}
    and (R \ge 0) and (S \ge N) } implies post?
{ post: (R^2 \le N \le (R+1)^2) and (R >= 0) }
```





Verification Via Backward Substitution (III)

Apply Axiom of Assignment and Rule of Consequence:

```
pre implies \{ (1 = 2*0 + 1) \text{ and } (1 = (0+1)^2) \}
   and (0^2 <= N) and (0>=0) }?
while S<=N loop { I: (T=2^*R+1) and (S=(R+1)^2) and (R^2<=N) and (R>=0) } R:=R+1:
```



Anthony Hall's Seven Myths of Formal Methods (I)

- Formal methods can guarantee that software is perfect
 - How do you ensure the initial spec is perfect?
- 2) Formal methods are all about program proving
 - They're also about modeling, communication, analyzing, demonstrating
- 3) Formal methods are only useful for safetycritical systems
 - Can be useful in any system



Anthony Hall's Seven Myths of Formal Methods (II)

- 4) Formal methods require highly trained mathematicians
 - Many methods involve nothing more than set theory and logic
- 5) Formal methods increase the cost of development
 - There is evidence that the opposite is true
- 6) Formal methods are unacceptable to users
 - When properly presented, users find them helpful



Anthony Hall's Seven Myths of Formal Methods (III)

- Formal methods are not used on real, large-scale software
 - They're used daily in many branches of industry



Bertrand Meyer's Seven Sins of the Specifier (I)

- 1) Noise
 - the presence in the specification text of an element that does not carry information relevant to any feature of the problem
 - Includes redundancy and remorse
- Silence
 - the existence of a feature of the problem that is not covered by any element of the text
- 3) Overspecification
 - the presence in the text of an element that corresponds not to a feature of the problem but to features of a possible solution



Bertrand Meyer's Seven Sins of the Specifier (II)

- 4) Contradiction
 - the presence in the text of two or more elements that define a feature of the system in an incompatible way
- 5) Ambiguity
 - the presence in the text of an element that makes it possible to interpret a feature of the problem in at least two different ways
- 6) Forward reference
 - the presence in the text of an element that uses features of the problem not defined until later in the text



Bertrand Meyer's Seven Sins of the Specifier (III)

- 7) Wishful thinking
 - the presence in the text of an element that defines a feature of the problem in such a way that a candidate solution cannot reasonably be validated



Limits to the Notion of "Correctness"

- Correctness of a program is always relative
 - It's relative to assumption that compiler is correct, which is relative to
 - Assumption that hardware architecture is correct, which is relative to
 - Assumption that digital approximations of continuous electromagnetic phenomena are correct, which is relative to
 - Assumption that the laws of physics are correct
- In other words, correctness is always a matter of demonstrating consistency of one spec with another, where the latter is assumed to be correct