

ICS 121 Topic 12: Formal Specifications

Introduction to Formal Methods and
Formal Specification
Types of Formal Specification
Abstract Model Specifications
State Transition Specifications
Algebraic Specifications
Axiomatic Specifications

What are Formal Methods?

- Formal Method (FM) =
 - specification language + formal reasoning
- Body of techniques supported by
 - precise mathematics
 - powerful analysis tools
- Rigorous effective mechanisms for system
 - modeling
 - synthesis
 - analysis

The diagram shows a flow from 'modeling' to 'Specification'. From 'Specification', there are two paths: one to 'Implementation' labeled 'synthesis', and one back to 'Specification' labeled 'analysis'. There is also a feedback loop from 'Implementation' back to 'Specification'.

Topic 12 Formal Specifications 2

Formal Specifications: what and why?

- Use of formalisms
 - e.g., logic, discrete mathematics, finite state machines
- To describe systems
 - e.g., system models, constraints, specifications, designs
- For broad range of effects
 - e.g., highly reliable, secure, safe systems and more effective production
- With varying levels of use
 - guidance: structuring what to say
 - documentation: unambiguous communication
 - rigor: formal specification and proofs
 - mechanisms: proof assistance, testing

Topic 12 Formal Specifications 3

Formal Specification in Software Development

- Formal specifications ground the software development process in the well-defined basis of computer science
- Orientation goes from customer to developer
- Formal specification expressed in language whose syntax and semantics are formally defined
 - hierarchical decomposition
 - mathematical foundation
 - graphical presentation
 - accompanied by informal description

Topic 12 Formal Specifications 4

Goals and Objectives

- Requirements Specification
 - clarify customer's requirements
 - reveal ambiguity, inconsistency, incompleteness
- System/Software Design
 - decomposition structural specifications of component relations and behavioral specification of components
 - refinement demonstrating that next level of abstraction satisfies higher level
- Verification
 - proving a realization satisfies its specification
- Validation
 - testing and debugging
- Documentation
 - communication between specifier, implementor, customer, clients

Topic 12 Formal Specifications 5

Specification and Design

The diagram shows three boxes in a sequence: 'Requirements-Analysis and Definition', 'Requirements-Specification', and 'Design'. Above the boxes, an arrow points right with the text: 'Increasing contractor involvement', 'Decreasing client involvement', and 'Increasing formalism possible'. Below the boxes, there are two shaded regions: 'SPECIFICATION' under the first two boxes and 'DESIGN' under the third box. Curved arrows indicate feedback loops from 'Requirements-Specification' back to 'Requirements-Analysis and Definition', and from 'Design' back to 'Requirements-Specification'.

Topic 12 Formal Specifications 6

Benefits of Using Formal Specifications and Methods

- higher quality software
- verifiability of implementation
- insight and understanding
- minimized maintenance and cost
- automated assistance
- simulation, animation, execution
- formal analysis
- guidance for testing
- transformation technology
- reduced liability and risks
- standards satisfaction

Topic 12 Formal Specifications 7

Formal Methods are not yet widely used

- Reasons
 - emerging technology with unclear payoff
 - little experience
 - lack of automated support
 - not especially user friendly
 - too many in-place techniques and tools
- Excuses
 - high learning curve
 - mathematical sophistication required
 - techniques not widely applicable
 - ignorance of advances

Topic 12 Formal Specifications 8

Desirable Properties of Formal Specifications

- Consistency
 - an implementation exists that satisfies a specification
- Complete (incrementally)
 - all required aspects are specified
- Unambiguous
 - all implementations that satisfy the specification are satisfactory (tradeoffs with completeness)
- Inference
 - consequence relation used to prove properties about the implementations that satisfy a specification

Topic 12 Formal Specifications 9

Types of Formal Specifications

- Behavioral specifications describe constraints on behavior of implementation – e.g.,
 - functionality
 - safety
 - security
 - performance
- Structural specifications describe constraints on internal composition of implementation
 - module interconnection
 - uses and is-composed-of
 - dependence relations

Topic 12 Formal Specifications 10

Characteristics of Specification & Support

- *Model-oriented* specifications
 - specify system behavior by constructing a model in terms of well-defined mathematical constructs
- *Property-oriented* specifications
 - specify system behavior in terms of properties that must be satisfied
- Visual specifications
 - specify system behavior and structure by graphical depictions
- Executable specifications
 - specify system behavior completely enough that specifications can run on a computer
- Tool support
 - syntactic checking
 - model checking
 - proof checking

Topic 12 Formal Specifications 11

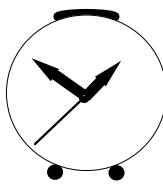
Basic Types of Behavioral Specification

- Abstract Model Specifications
 - defines operations in terms of well-defined mathematical model
- Algebraic Specifications
 - defines operations by a collection of equivalence relations
- State Transition Specifications
 - defines operations in terms of states and transitions
- Axiomatic Specifications
 - defines operations by logical assertions

Topic 12 Formal Specifications 12

Formal Specification Languages: Clock Example

- + Initially, the time is midnight, the bell is off, and the alarm is disabled.
- + Whenever the current time is the same as the alarm time and the alarm is enabled, the bell starts ringing.
This is the only condition under which the bell begins to ring.
- + The alarm time can be set at any time.
- + Only when the alarm is enabled can it be disabled.
- + If the alarm is disabled while the bell is ringing, the bell stops ringing.
- + Resetting the clock and enabling or disabling the alarm are considered to be done instantaneously.



Topic 12 Formal Specifications 13

Abstract Model Specifications

- Explicitly describes behavior in terms of a model using well-defined types (sets, sequences, relations, functions) and defines behavior by showing effects on model
 - State is explicit in the model
 - Objects can be built hierarchically
- Specification includes
 - type: syntax of object being specified
 - model: underlying structure
 - invariant: properties of modeled object
 - pre/post conditions: semantics of operations
- Pros and Cons
 - may suggest an implementation
 - widely applicable because of modeling orientation
- Various notations: VDM, Z, RAISE

Topic 12 Formal Specifications 14

Abstract Model Specifications: Process using Z

- Specification using Z abstract model specifications
 - 1. establish the object schemas (objects and attributes)
 - including invariant properties of objects
 - 2. establish the operation schemas
 - schema modifiability (Δ vs. E schema)
 - operation signatures (input? vs. output!)
 - state modifications (attribute vs. attribute')

Topic 12 Formal Specifications 15

Abstract Model Specifications: the Z Notation

specification name [generic parameters] _____

signature _____

predicate _____

- a Z specification is a collection of schemas
- a schema introduces some entities and invariant properties
- the signature may make a defined schema visible
- the schema signature defines each entity's name and type (syntax)
- the predicate defines the relationships between the entities that must always hold (semantics)
- should be supported by informal description

Topic 12 Formal Specifications 16

Abstract Model Specifications: Z Clock

BellStatus: {quiet, ringing}, AlarmStatus: {disabled, enabled}

Clock

time, alarm_time: N

bell: BellStatus

alarm: AlarmStatus

InitClock

Δ Clock

$(time' = midnight) \wedge (bell' = quiet) \wedge (alarm' = disabled)$

Tick

Δ Clock

$(time' = succ(time))$
 $(alarm_time' = time') \wedge (alarm' = enabled) \Rightarrow (bell' = ringing)$
 $(\neg(alarm_time' = time') \wedge (alarm' = enabled)) \Rightarrow (bell' = bell)$

Topic 12 Formal Specifications 17

Abstract Model Specifications: Z Clock, continued

SetAlarmTime

Δ Clock

new_time?: N

$(alarm_time' = new_time?)$
 $((alarm_time' = time') \wedge (alarm' = enabled) \Rightarrow (bell' = ringing))$
 $(\neg(alarm_time' = time') \wedge (alarm' = enabled)) \Rightarrow (bell' = bell)$
 $(time' = time) \wedge (alarm' = alarm)$

EnableAlarm

Δ Clock

$(alarm = disabled) \Rightarrow (alarm' = enabled) \wedge$
 $(alarm_time' = time') \Rightarrow (bell' = ringing) \wedge$
 $(\neg(alarm_time' = time')) \Rightarrow (bell' = bell) \wedge$
 $(time' = time) \wedge (alarm_time' = alarm_time)$

DisableAlarm

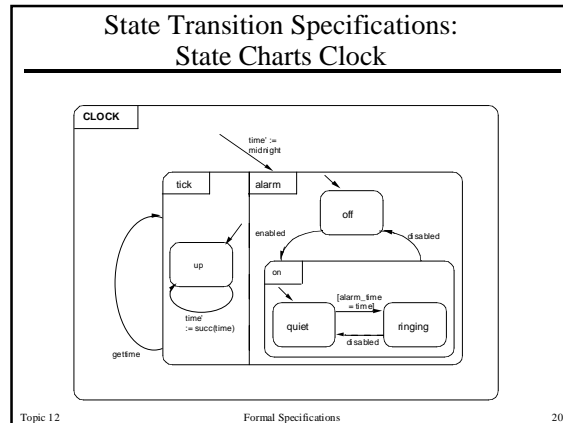
Δ Clock

Topic 12 Formal Specifications 18

State Transition Specifications

- Explicitly describes system behavior by a set of states and defines operations as transitions between states or observations on state
- Specification includes
 - states: possible values
 - transitions: semantics by state transformations and observations
- Pros and Cons
 - free of representational details (except augmentations)
 - state explosion is common
 - extensions to minimize states and modularize
 - particularly applicable to control systems, languages, hardware
- Graphical as well as textual notations: StateCharts, ASLAN, Paisley, InaJo, Special

Topic 12 Formal Specifications 19



Algebraic Specification

- Objects specified as algebraic sorts in terms of equivalence relations between associated operations
 - State is concealed in objects
 - Objects can be built hierarchically
- Specification includes
 - functionality: syntax and legal constructions
 - relations: semantics by equivalence classes
- Pros and Cons
 - only pure functions described (no side effects)
 - supports extensibility of data abstractions
 - often hard to comprehend and construct
- Various notations: OBJ, Larch, Clear, Anna

Topic 12 Formal Specifications 21

Algebraic Specifications: Process

- Specification using algebraic sorts
 1. establish the sorts (objects and attributes)
 - constructor operations
 - access operations
 2. establish the necessary operations
 3. establish the equivalence relations
 - rule of thumb: a relation for each access over each constructor
 - simplified when constructors defined in terms of imports

Topic 12 Formal Specifications 22

Algebraic Specifications: a simple notation

specification name (generic parameters)
sort name
imports list of specification names
operation signatures
relations

- an algebraic specification is a collection of sorts
- a sort specifies an object class (or abstract data type)
- importing specifications makes their defined sorts visible
- the operation signatures define each operation's name and the sorts of parameters and results (syntax)
- the relations define the effect of applying operations (semantics)
- should be supported by informal description

Topic 12 Formal Specifications 23

Algebraic Specifications: Algebraic CLOCK

operation signatures

- init: \rightarrow CLOCK
- tick: CLOCK \rightarrow CLOCK
- setalarm: CLOCK x TIME \rightarrow CLOCK
- enable: CLOCK \rightarrow CLOCK
- disable: CLOCK \rightarrow CLOCK
- time: CLOCK \rightarrow TIME
- alarm_time: CLOCK \rightarrow TIME
- bell: CLOCK \rightarrow {ringing, quiet}
- alarm: CLOCK \rightarrow {on, off}

Topic 12 Formal Specifications 24

Algebraic Specifications: Algebraic CLOCK, *continued*

relations

- time(init) -> midnight
- time(tick(C)) -> time(C) + 1
- time(setalarm(C,T)) -> time(C)
- time(enable(C)) -> time(C)
- time(disable(C)) -> time(C)

- alarm_time(init) -> midnight
- alarm_time(tick(C)) -> alarm_time(C)
- alarm_time(setalarm(C,T)) -> T
- alarm_time(enable(C)) -> alarm_time(C)
- alarm_time(disable(C)) -> alarm_time(C)

Topic 12 Formal Specifications 25

Algebraic Specifications: Algebraic CLOCK, *continued*

axioms

```

bell(init) -> quiet
bell(tick(C)) -> (if alarm_time(tick(C)) = time(tick(C)) and alarm(C) = on
then ringing else quiet)
bell(setalarm(C,T)) -> (if T = time(C) and alarm(C) = on
then ringing else quiet)
bell(enable(C)) -> (if alarm_time(C) = time(tick(C))
then ringing else quiet)
bell(disable(C)) -> quiet

alarm(init) -> off
alarm(tick(C)) -> alarm(C)
alarm(setalarm(C,T)) -> alarm(C)
alarm(enable(C)) -> (if alarm(C) = off then on)
alarm(disable(C)) -> (if alarm(C) = on then off)
    
```

Topic 12 Formal Specifications 26

Axiomatic Specifications

- Implicitly defines behavior in terms of [first-order] logic formulas specifying input/output assertions
- Specification includes
 - operation interfaces: input/output parameters
 - operation axioms: pre/post assertions on input/output
 - possibly also intermediate assertions
- Pros and Cons
 - fairly easy to understand
 - widely applicable (although hard to scale up)
 - most widely used technique in proving (inductive assertion method)
 - foundation of mathematics in software development
- Many languages support this type of specification:
 - VDM, Anna
 - Extensions include various logics for specific application domains (e.g., temporal logic: RTIL, GIL)

Topic 12 Formal Specifications 27

Axiomatic Specifications: VDM Clock

INIT()

```

ext wr time: N, bell: {quiet, ringing}, alarm: {disabled, enabled}
pre true
post (time' = midnight) ^ (bell' = quiet) ^ (alarm' = disabled)
    
```

TICK()

```

ext wr time: N, bell: {quiet, ringing}
rd alarm_time: N, alarm: {disabled, enabled}
pre true
post (time' = succ(time)) ^ (if (alarm_time' = time') ^ (alarm' = enabled)
then (bell' = ringing) else (bell' = bell))
    
```

Topic 12 Formal Specifications 28

Axiomatic Specifications: VDM Clock, *continued*

```

SETALARMTIME(new_time: N)
ext wr alarm_time: N, bell: {quiet, ringing}
rd time: N, alarm: {disabled, enabled}
pre true
post (alarm_time' = new_time) ^ (if (alarm_time' = time') ^ (alarm' = enabled)
then (bell' = ringing) else (bell' = bell))

ENABLEALARM()
ext wr alarm: {disabled, enabled}, bell: {quiet, ringing}
rd time: N, alarm_time: N
pre alarm = disabled
post (alarm' = enabled) ^ (if (alarm_time' = time')
then (bell' = ringing) else (bell' = bell))

DISABLEALARM()
ext wr alarm: {disabled, enabled}, bell: {quiet, ringing}
pre alarm = enabled
post (alarm' = disabled) ^ (bell' = quiet)
    
```

Topic 12 Formal Specifications 29

Axiomatic Specifications: GIL Clock

Initialization: initially the alarm is disabled and the bell is off

Won't Ring: the bell is off from disabled to (enabled and time=alarm)

Will Ring: the bell is on from (enabled and time=alarm) to disabled

Topic 12 Formal Specifications 30

Something to think about:

Formal Specification Comparison

- Formal Specification Languages differ in the qualities and application domains for which they are best suited
- Axiomatic Specifications
- State Transition Specifications
- Abstract Model Specifications
- Algebraic Specifications

Topic 12

Formal Specifications

31