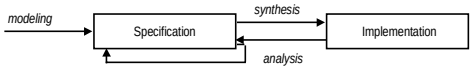
ICS 121 Lecture Notes

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



What are Formal Methods?

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

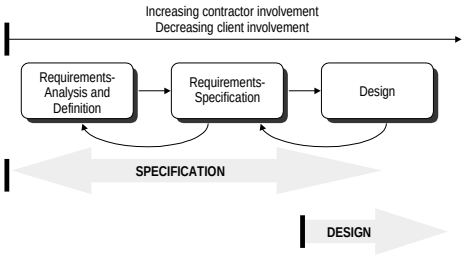
Formal Specification in Software Development

- Formal specifications ground the software development process in the well-defined basis of computer science
- Emphasis switches 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

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 specificand (implementation) satisfies its specification
- Validation
  - testing and debugging
- Documentation
  - communication between specifier, implementor, customer, clients

Specification and Design



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
- standard satisfaction

Topic 10
Formal Methods

ICS 121 Lecture Notes

Formal Specifications 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

Formal Specification Languages

- A formal specification language consists of
  - *syntax* (the notation)
  - *semantics* (the meaning)
  - *satisfies* (relation defining which objects satisfy which notations)
- A formal specification defines
  - syntax (signature of the mapping)
  - semantics (meaning of the mapping)
  - exceptions (undefined/erroneous mappings)
- If sat(syn,sem) then
  - syn is a *specification* of sem
  - sem is a *specificand* of syn

Desirable Properties

- Consistency
  - a specificand exists that satisfies a specification
- *Completeness* (incrementally)
  - all aspects of specificands are specified
- Unambiguous
  - exactly one specificand satisfies a specification (may be a set if specification is not complete)
- Inference
  - consequence relation used to prove properties about the specificands that satisfy a specification

Types of Formal Specifications

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

Characteristics

- *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

Basic Specification Language Types

❶ Abstract Model Specifications
- defines operations in terms of well-defined mathematical model

❷ Algebraic Specifications
- defines operations by a collection of equivalence relations

❸ State Transition Specifications
- defines operations in terms of states and transitions

❹ Axiomatic Specifications
- defines operations by logical assertions

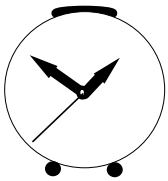
Topic 10 Formal Methods

ICS 121 Lecture Notes

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.



❶ Abstract Model Specifications

- **Explicitly describes behavior in terms of a model using well-defined types (sets, sequences, relations, functions) and defines operations by showing effects on model**
- **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**
  - state is made explicit in model
  - suggests an implementation
  - widely applicable because of modeling orientation
- **Various notations: VDM, Z, RAISE**

Abstract Model Specifications

- **Objects specified by effect of operations on a model in terms of well-understood mathematical entities (e.g., sets, sequences, relations, functions)**
- **State is explicit in the model**
- **Objects can be built hierarchically**
- **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 ($^{\Delta}$ vs. E schema)
    - operation signatures (input? vs. output!)
    - state modifications (attribute vs. attribute')

Abstract Model Specifications: the Z Notation


```
┌─ specification name [generic parameters] ──────
│ signature
├────────────────────────────────────────
│ predicate
└────────────────────────────────────────
```

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

Abstract Model Specifications: Z Clock - 1

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

```
┌─ Clock ──────────────────────────
│ time, alarm_time: N
│ bell: BellStatus
│ alarm: AlarmStatus
└──────────────────────────────────

┌─ InitClock ──────────────────────
│ Δ Clock
├──────────────────────────────────
│ (time′ = midnight) ∧ (bell′ = quiet) ∧ (alarm′ = disabled)
└──────────────────────────────────

┌─ Tick ───────────────────────────
│ Δ Clock
├──────────────────────────────────
│ (time′ = succ(time))
│ (alarm_time′ = time′) ∧ (alarm′ = enabled) => (bell′ = ringing)
│ (~((alarm_time′ = time′) ∧ (alarm′ = enabled)) => (bell′ = bell)
│ (alarm′ = alarm) ∧ (alarm_time′ = alarm_time)
└──────────────────────────────────
```

Abstract Model Specifications: Z Clock - 2


```
┌─ SetAlarmTime ───────────────────
│ Δ Clock
│ new_time?: N
├──────────────────────────────────
│ (alarm_time′ = new_time?)
│ ((alarm_time′ = time′) ∧ (alarm′ = enabled) => (bell′ = ringing)
│ (~((alarm_time′ = time′) ∧ (alarm′ = enabled)) => (bell′ = bell)
│ (time′ = time) ∧ (alarm′ = alarm)
└──────────────────────────────────

┌─ EnableAlarm ────────────────────
│ Δ Clock
├──────────────────────────────────
│ (alarm = disabled) => (alarm′ = enabled) ∧
│       ( (alarm_time′= time′) => (bell′ = ringing) ∧
│       (~(alarm_time′ = time′)) => (bell′ = bell) ) ∧
│       (time′ = time) ∧ (alarm_time′ = alarm_time)
└──────────────────────────────────

┌─ DisableAlarm ───────────────────
│ Δ Clock
├──────────────────────────────────
│ (alarm = enabled) => (alarm′ = disabled) ∧ (bell′ = quiet) ∧
│       (time′ = time) ∧ (alarm′ = alarm) ∧ (alarm_time′ = alarm_time)
└──────────────────────────────────
```

Topic 10 Formal Methods

❷ 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 using algebraic sorts
  1. establish the sorts (objects and attributes)
  2. establish the necessary operations
     - constructor operations
     - access 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

Algebraic Specifications - 2

- 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
  - particularly applicable to abstract data types
- Various notations: OBJ, Larch, Clear, Anna

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

Algebraic Specifications: Algebraic CLOCK

<u>operation signatures</u>

```
init: –> CLOCK
tick: CLOCK –> CLOCK
setalarm: CLOCK x TIME –> CLOCK
enable: CLOCK –> CLOCK
disable: CLOCK –> CLOCK
time: CLOCK –> TIME
alarm_time: CLOCK –> TIME
bell: CLOCK –> {ringing, quiet}
alarm: CLOCK –> {on, off}
```


Algebraic Specifications: Algebraic CLOCK - 2

<u>relations</u>

```
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)
```


Algebraic Specifications: Algebraic CLOCK - 3

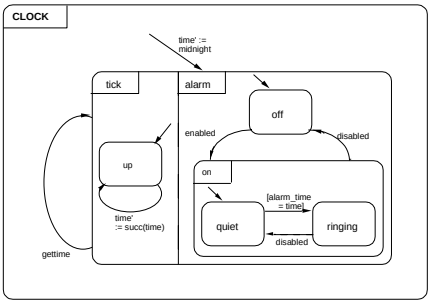

```
bell(init) –> quiet
bell(tick(C)) –> (if alarm_time(tick(C)) = time(tick(C)) and alarm(C) = on
                 then ringing else quiet)
bell(setalarm(C,T)) –> (if T = time(C) and alarm(C) = on
                       then ringing else quiet)
bell(enable(C)) –> (if alarm_time(C) = time(tick(C))
 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

ICS 121 Lecture Notes

❸ 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

State Transition Specifications: State Charts Clock



State Transition Specifications: ASLAN Clock


```
SPECIFICATION Clock

LEVEL Top_Level

TYPE BellStatus IS (quiet, ringing), AlarmStatus IS (disabled, enabled)

VARIABLE time, alarm_time: INTEGER, bell: BellStatus, alarm: AlarmStatus

INITIAL (time = midnight) & (bell = quiet) & (alarm = disabled)

INVARIANT  TRUE

TRANSITION tick
   ENTRY  TRUE

   EXIT (time = succ(time′))
         & (if ((alarm_time = time) & (alarm = enabled))
            then (bell = ringing)  else (bell = bell′)  fi)
         & (alarm = alarm′) & (alarm_time = alarm_time′)
```


State Transition Specifications: ASLAN Clock - 2


```
TRANSITION set_alarm (new_time: INTEGER)
   ENTRY  TRUE
   EXIT (alarm_time = new_time)
         & (if ((alarm_time = time) & (alarm = enabled))
            then (bell = ringing)  else (bell = bell′)  fi)
         & (time = time′) & (alarm = alarm′)

TRANSITION enable_alarm
   ENTRY  alarm = disabled
   EXIT (alarm = enabled)
         & (if (alarm_time = time)
            then (bell = ringing)  else (bell = bell′)  fi)
         & (time = time′) & (alarm_time = alarm_time′)

TRANSITION disable_alarm
   ENTRY  alarm = enabled
   EXIT (alarm = disabled)  & (bell = quiet)

END Top_Level

END Clock
```


❹ Axiomatic Specifications

- Implicitly defines behavior in terms of [first-order] logic formulas specifying input/output assertions (and possibly intermediate assertions)
- Specification includes
  - operation interfaces: input/output parameters
  - operation axioms: pre/post assertions on input/output
- 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)

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 10 Formal Methods

ICS 121 Lecture Notes

Axiomatic Specifications: VDM Clock - 2


```
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 10
Formal Methods