Lecture 17: Formal Modeling Methods

Formal Modeling Techniques

- Definition of FM
- Why use FM?

Program Specification vs. Requirements Modeling

Example Formal Methods:

- RSML
- SCR
- RML
- Telos
- Albert II

Tips on formal modeling

What are Formal Methods?

Broad View (Leveson)

- application of discrete mathematics to software engineering
- involves modeling and analysis
- with an underlying mathematically-precise notation

Narrow View (Wing)

- Use of a formal language
  - a set of strings over some well-defined alphabet, with rules for distinguishing which strings belong to the language
- Formal reasoning about formulae in the language
  - E.g. formal proofs: use axioms and proof rules to demonstrate that some formula is in the language

For requirements modeling…

- A notation is formal if:
  - …it comes with a formal set of rules which define its syntax and semantics.
  - …the rules can be used to analyse expressions to determine if they are syntactically well-formed or to prove properties about them.

Formal Methods in Software Engineering

What to formalize?

- models of requirements knowledge (so we can reason about them)
- specifications of requirements (so we can document them precisely)
- Specifications of program design (so we can verify correctness)

Why formalize?

- Removes ambiguity and improves precision
- To verify that the requirements have been met
- To reason about the requirements/designs
  - Properties can be checked automatically
  - Test for consistency, explore consequences, etc.
- To animate/execute specifications
  - Helps with visualization and validation
- …because we have to formalize eventually anyway
  - Need to bridge from the informal world to a formal machine domain

Why people don't formalize!

- Formal Methods tend to be lower level than other techniques
  - They include too much detail
- Formal Methods concentrate on consistent, correct models
  - …most of the time your models are inconsistent, incorrect, incomplete…
- People get confused about which tools are appropriate:
  - specification of program behaviour vs. modeling of requirements
  - formal methods advocates get too attached to one tool!
- Formal methods require more effort
  - ...and the payoff is deferred

Varieties of formal analysis

Consistency analysis and typechecking

- "Is the formal model well-formed?"
- Assumes "well-formedness" of the model corresponds to something useful…

Validation:

- Animate the model on small examples
- Formal challenges:
  - "if the model is correct then the following property should hold..."
- 'what if' questions:
  - reasoning about the consequences of particular requirements;
  - reasoning about the effect of possible changes

Predicting behavior

- State exploration (E.g. through model checking)
- Checking application properties:
  - "will the system ever do the following..."

Verifying design refinement

- "does the design meet the requirements?"

Three traditions …

Formal Specification Languages

Grew out of work on program verification

Spawned many general purpose specification languages

- Good for specifying the behaviour of program units

Key technologies: Type checking, Theorem proving

Applicable to program design
- closely tied to program semantics

Examples: Larch, Z, VDM, …

Reactive System Modelling

Formalizes dynamic models of system behaviour

Good for reactive systems (e.g. real-time, embedded control systems)

- can reason about safety, liveness, performance(?)

Key technologies: Consistency checking, Model checking

Applicable to Requirements
- Languages developed specifically for RE

Examples: Statecharts, RSML, Parnas-tables, SCR, …

Formal Conceptual Modelling

For capturing real-world knowledge in RE

Focuses on modelling domain entities, activities, agents, assertions, goals,…

- use first order predicate logic as the underlying formalism

Key technologies: inference engines, default reasoning, KBS-shells

Applicable to Requirements
- Capture key requirements concepts

Examples: Reqts Apprentice, RML, Telos, Albert II, …

(1) Formal *Specification* Languages

Three basic flavours:

- Operational - specification is executable abstraction of the implementation
  - good for rapid prototyping
  - e.g., Lisp, Prolog, Smalltalk
- State-based - views a program as a (large) data structures whose state can be altered by procedure calls…
  - … using pre/post-conditions to specify the effect of procedures
  - e.g., VDM, Z
- Algebraic - views a program as a set of abstract data structures with a set of operations…
  - … operations are defined declaratively by giving a set of axioms
  - e.g., Larch, CLEAR, OBJ

Developed for specifying *programs*

- Programs are formal, man-made objects
  - … and can be modeled precisely in terms of input-output behaviour
- These languages are NOT appropriate for requirements modeling
  - requirements specification ≠ program specification

(2) Reactive System *Modelling*

Modeling how a system should behave

- General approach:
  - Model the environment as a state machine
  - Model the system as a state machine
  - Model safety, liveness properties of the machine as temporal logic assertions
  - Check whether the properties hold of the system interacting with its environment

Examples:

- Statecharts
  - Harel's notation for modeling large systems
  - Adds parallelism, decomposition and conditional transitions to STDs
- RSML
  - Heimdahl & Leveson's Requirements State Machine Language
  - Adds tabular specification of complex conditions to Statecharts
- A7e approach
  - Major project led by Parnas to formalize A7e aircraft requirements spec
  - Uses tables to specify transition relations & outputs
- SCR
  - Heitmeyer et. al. "Software Cost Reduction"
  - Extends the A7e approach to include dictionaries & support tables

(3) Formal Conceptual *Modelling*

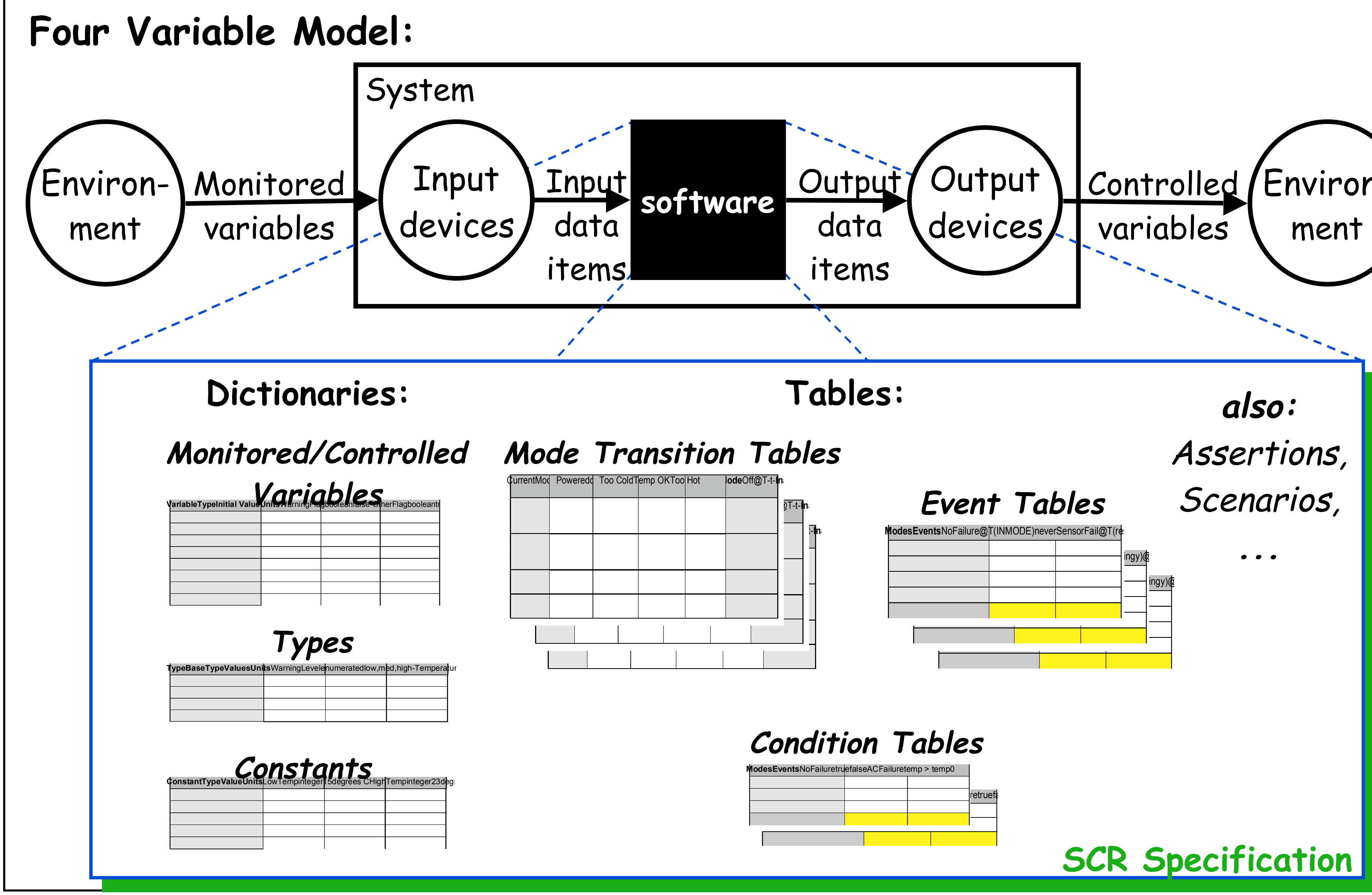
General approach

- model the world beyond software functions
  - build models of humans' knowledge/beliefs about the world
  - draws on techniques from AI and Knowledge Representation
- make use of abstraction & refinement as structuring primitives

Examples:

- RML - Requirements Modeling Language
  - Developed by Greenspan & Mylopoulos in mid-1980s
  - First major attempt to use knowledge representation techniques in RE
  - Object oriented language, with classes for activities, entities and assertions
  - Uses First Order Predicate Language as an underlying reasoning engine
- Telos
  - Extends RML by creating a fully extensible ontology
  - meta-level classes define the ontology (the basic set is built in)
- Albert II
  - developed by Dubois & du Bois in the mid-1990s
  - Models a set of interacting agents that perform actions that change their state
  - uses an object-oriented real-time temporal logic for reasoning

Example: SCR


Four Variable Model:
System
Environ-ment
Monitored variables
Input devices
Input data items
software
Output data items
Output devices
Controlled variables
Environ-ment
Dictionaries:
Monitored/Controlled Variables
Types
Constants
Tables:
Mode Transition Tables
Event Tables
Condition Tables
also:
Assertions,
Scenarios,
...
SCR Specification

SCR basics

Source: *Adapted from Heitmeyer et. al. 1996.*

Modes and Mode classes

- A mode class is a finite state machine, with states called *system modes*
  - Transitions in each mode class are triggered by *events*
- Complex systems are described using a number of mode classes operating in parallel

System State

- A (system) state is defined as:
  - the system is in exactly one mode from each mode class…
  - …and each variable has a unique value

Events

- An event occurs when any system entity changes value
  - An *input event* occurs when an *input* variable changes value
  - Single input assumption - only one input event can occur at once
  - Notation: @T(c) means "c changed from false to true"
- A conditioned event is an event with a predicate
  - @T(c) WHEN d means: "c became true when c *was* false and d *was* true"

SCR Tables

Source: *Adapted from Heitmeyer et. al. 1996.*

Mode Class Tables

- Define the set of *modes* (states) that the software can be in.
- A complex system will have many different modes classes
  - Each mode class has a mode table showing the conditions that cause transitions between modes
- A mode table defines a *partial function* from modes and events to modes

Event Tables

- An event table defines how a term or controlled variable changes in response to input events
- Defines a *partial function* from modes and events to variable values

Condition Tables

- A condition table defines the value of a term or controlled variable under every possible condition
- Defines a *total function* from modes and conditions to variable values

Example: Temp Control System

Source: *Adapted from Heitmeyer et. al. 1996.*

Mode transition table:

Current Mode	Powered on	Too Cold	Temp OK	Too Hot	**New Mode**
Off	@T	-	t	-	**Inactive**
	@T	t	-	-	**Heat**
	@T	-	-	t	**AC**
Inactive	@F	-	-	-	**Off**
	-	@T	-	-	**Heat**
	-	-	-	@T	**AC**
Heat	@F	-	-	-	**Off**
	-	-	@T	-	**Inactive**
AC	@F	-	-	-	**Off**
	-	-	@T	-	**Inactive**

Failure modes

Source: *Adapted from Heitmeyer et. al. 1996.*

Mode transition table:

Current Mode	Powered on	Cold Heater	Too Cold	Warm AC	Too Hot	New Mode
NoFailure	t	@T	t	-	-	**HeatFailure**
	t	-	-	@T	t	**ACFailure**
HeatFailure	t	@F	t	-	-	**NoFailure**
ACFailure	t	-	-	@F	t	**NoFailure**

Event table:

Modes		
NoFailure	@T(INMODE)	never
ACFailure, HeatFailure	never	@T(INMODE)
Warning light =	Off	On

Using Formal Methods

Selective use of Formal Methods

- Amount of formality can vary
- Need not build complete formal models
  - Apply to the most critical pieces
  - Apply where existing analysis techniques are weak
- Need not formally analyze every system property
  - E.g. check safety properties only
- Need not apply FM in every phase of development
  - E.g. use for modeling requirements, but don't formalize the system design
- Can choose what level of abstraction (amount of detail) to model

Lightweight Formal Methods

- Have become popular as a means of getting the technology transferred
- Two approaches
  - Lightweight *use of* FMs - selectively apply FMs for partial modeling
  - Lightweight FMs - new methods that allow unevaluated predicates

References

van Vliet, H. "Software Engineering: Principles and Practice (2nd Edition)" Wiley, 1999.

van Vliet gives a good introduction to formal methods in chapter 15. In particular, sections 15.1 and 15.5 are worth reading, to give a feel for the current state of the art, and the problems that hinder the use of formal methods in practice. van Vliet describes a completely different set of formal modeling techniques from those covered in this lecture – he concentrates on methods that can be used for program design models, rather than requirements models.

Heitmeyer, C. L., Jeffords, R. D., & Labaw, B. G. (1996). Automated Consistency Checking of Requirements Specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3), 231-261.

Describes SCR in detail.