Formal Specification Techniques

CAS 707

McMaster University, Winter 2016

Wolfram **Kahl**

kahl@cas.mcmaster.ca

6 January 2016

Goals

- Understanding of the motivation of mathematical approaches to software specification
- Knowledge of typical approaches to formal software specification and verification
- Ability to produce and evaluate formal software specifications
- Experience with a selection of current software verification tools
- Knowlege of different logical formalisms, of the principles of related tool support, and associated selection criteria

Learning Objectives

- **Precondition:** What you are expected to have learnt before taking this course
- **Postcondition:** What you will be expected to have learnt at the end of this course

Learning Objectives — Postcondition (1)

Students should **know and understand**

1. Big-step operational semantics of a simple imperative programming language
2. Hoare logic proof rules for a simple imperative programming language
3. Verification condition generation for a simple imperative programming language
4. Scope and limitations of automated verification, proof, and program analysis tools.

Learning Objectives — Postcondition (2)

Students should **know and understand**

1. Theory and applications of algebraic specification
2. The spectrum of temporal logics
3. Patterns of temporal-logics specifications, especially safety and lifeness conditions
4. The principles behind model checking of temporal-logics specifications.

Learning Objectives — Postcondition (3)

Students should **be able to**

1. Translate English specifications of program fragments into formal pre- and post-condition specifications.
2. Produce counterexample traces using operational semantics.
3. Annotate their programs with appropriate specifications and assertions for mechanised analysis with at least one verification tool.
4. Use Hoare logic to prove partial and total correctness of simple imperative programs.
5. Use VCGen algorithms to extract verification conditions from programs in a simple imperative programming language.

Learning Objectives — Postcondition (4)

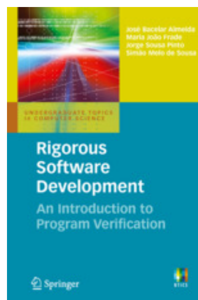
Students should **be able to**

1. Understand structured algebraic specifications
2. Produce algebraic specifications with appropriate structure
3. Produce state-based models e.g. in Z or B, and perform model refinements.
4. Produce state transition models and temporal-logic specifications for (components of) concurrent systems, and verify them using model checking software

Rough Outline

- Specifying and verifying C Programs — **Frama-C** *(RSD chapter 9, and parts of RSD chapters 5-8)*
- Review of Predicate Logic *(parts of RSD chapter 4)*
- Hoare logic and VCGen in depth — **Frama-C, AltErgo, . . .** *(RSD chapters 5–8, 10)*
- Algebraic Specification — **Maude, CASL**
- Specification and verification of dynamic systems: Rewriting logic, temporal logic, model-checking — **Maude, Spin, nusmv, . . .**
- Separation logic, fully certified verification — **VST, CompCert, Coq**
- Selection of other topics, e.g.: State-based modelling (Z, B), Specification via types, JML

Partial Textbook



“RSD”: **Rigorous Software Development — An Introduction to Program Verification**, by José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, and Simão Melo de Sousa. Springer, London, 2011.

(available electronically via the McMaster library).

More Literature

“Using Z”: **Using Z: Specification, Refinement, and Proof**, Jim Woodcock and Jim Davies, Prentice Hall, 1996 (out of print; available on-line)

“Huth-Ryan”: **Logic in Computer Science, Modelling and Reasoning about Systems**, Michael R. A. Huth and Mark D. Ryan, Cambridge University Press, 2nd edition 2004.

“Alagar-Periyasami”: **Specification of Software Systems**, V. S. Alagar and K. Periyasamy, Springer 2011 (available electronically via the McMaster library).

Covers all topics of this course to some degree; lots of explanations, but frequently lacks precision.

. . . To be announced . . .

Grading

Item		Weight
Assignments (graded only summarily)	— together:	**25%**
1 Project: Handout, Presentation, Source files		**15%**
1 Oral Midterm (5–10 min.)		**15%**
1 Attendance and participation		**5%**
Oral Final Exam (≈20 min)		**40%**
	=	**100%**

Validation versus Verification

- **Validation** asks:

  Are we building the right product?

- **Verification** asks:

  Are we building the product right?

Necessary Ingredients of a Formal Method

- Formal syntax for the specification language

  (Tools can check syntactic correctness)
- Formal semantics for the specification language

  (No disagreement about the meaning of specifications possible)
- Formal reasoning about the specification language

  (Correctness can be mechanically verified)

  (Finding proofs may still require human assistance, but proofs can be checked mechanically.)

Proof systems

- Formal System = Deductive System
- Antagonism:

  logical expressivity
  versus
  automation of the deduction
- That is:
  - "If you want automatic proofs, don't hope to be able to specify all interesting properties"
  - "If you want to be able to specify all interesting properties, don't hope for automatic proofs"

A formal development example done in B

> *First real success was Meteor line 14 driverless metro in Paris: Over 110 000 lines of B models were written, generating 86 000 lines of Ada. No bugs were detected after the proofs, neither at the functional validation, at the integration validation, at on-site test, nor since the metro lines operate (October 1998). The safety-critical software is still in version 1.0 in year 2007, without any bug detected so far.*

In Formal Methods in Safety-Critical Railway Systems, Thierry Lecomte, Thierry Servat, Guilhem Pouzancre.

ASTRÉE Success story (excerpt from its web-site)

The development of ASTRÉE started from scratch in Nov. 2001. Two years later, the main applications have been the static analysis of synchronous, time-triggered, real-time, safety critical, embedded software written or automatically generated in the C programming language. ASTRÉE has achieved the following unprecedented results:

- In Nov. 2003, ASTRÉE was able to prove completely automatically the absence of any RTE in the primary flight control software of the Airbus A340 fly-by-wire system, a program of 132,000 lines of C analyzed in 1h20 on a 2.8 GHz 32-bit PC using 300 Mb of memory (and 50mn on a 64-bit AMD Athlon 64 using 580 Mb of memory).
- From Jan. 2004 on, ASTRÉE was extended to analyze the electric flight control codes then in development and test for the A380 series. The operational application by Airbus France at the end of 2004 was just in time before the A380 maiden flight on Wednesday, 27 April, 2005.
- In April 2008, ASTRÉE was able to prove completely automatically the absence of any RTE in a C version of the automatic docking software of the Jules Vernes Automated Transfer Vehicle (ATV) enabling ESA to transport payloads to the International Space Station.

Learning Objectives — Precondition (1)

- **Syntax of propositional logic**

  What are the pieces called you use to write down a propositional-logic formula?

Learning Objectives — Precondition (2)

- **Syntax of predicate logic, including variable binding issues**

  How are predicate-logic formulae different from propositional-logic formulae?

  Show how naïve substitution can lead to problems, and how these are avoided.

Learning Objectives — Precondition (3)

- **Semantics of propositional logic**

  What is a tautology? — contradiction? — contingency?

  How do you find out whether a propositional logic formula is a tautology?

  Other methods?

Learning Objectives — Precondition (4)

- **Semantics of predicate logic**

  What is a structure?

  What is validity? — satisfiability?

Learning Objectives — Precondition (5)

- **At least one proof system for predicate logic**

Learning Objectives — Precondition (6)

- **Basics of the metatheory of predicate logic**

  What is soundness?

  What is completeness?

  (Can you state the compactness theorem?)

Learning Objectives — Precondition (7)

- **Principles of typed expressions, and the types of the operators they are using**

  Formalise: x or y is greater than 5.

  Discuss.

Learning Objectives — Precondition (8)

- **Principles of calculational proofs**

  Why/when/for what is something in the shape of the following a proof?

$$\begin{array}{ll} & E_1 \\ = & \langle Q_1 = Q_2 \rangle \\ & E_2 \\ < & \langle R_1 < R_2 \rangle \\ & E_3 \end{array}$$

  How about:

$$\begin{array}{ll} & E_1 \\ \Leftrightarrow & \langle Q_1 = Q_2 \rangle \\ & E_2 \\ \Rightarrow & \langle R_1 \Rightarrow R_2 \rangle \\ & E_3 \end{array}$$

Learning Objectives — Precondition (9)

- **Basic concepts and theorems about sets, functions and (especially binary) relations**

  Define:
  - symmetric
  - equivalence
  - order
  - injective
  - transitive closure

Learning Objectives — Precondition (10)

- **Standard kinds of algebras**

  What is a monoid? group? ring? field? vector space?

  What is a monoid homomorphism?

  What is a graph? lattice?

  What is a graph homomorphism? lattice homomorphism?

Learning Objectives — Precondition (11)

- **Imperative programming**

  Write an iterative program fragment that stores the factorial of n in r.

  Explain linear/cascading/nested/tail recursion.

  Write an iterative program fragment that stores the n-th Fibonacci number in r.

Learning Objectives — Precondition (12)

- **Basic datastructures and algorithms**

  When do you use doubly-linked lists?

  Write a datatype declaration and an insert function for ordered binary trees.

  Write a datatype declaration and a delete function for some kind of balanced ordered binary trees.

Learning Objectives — Precondition (13)

- **Basics of functional programming**

  Declare and define `map` and `foldRight` over lists.

  State and prove (by structural induction) some simple properties of `map` and `foldRight`

  What exactly is a higher-order function?

Learning Objectives — Precondition (14)

- **Basic concepts of decidability, computability, and complexity**

  If we say “P is decidable”, what kind of entity is P?

  If we say “P is decidable”, what does this mean?

  Give an example for a function that is not computable!

Learning Objectives — Precondition (15)

- **Common software development process models**

  Name and describe "the archetypal software development process model"

  Name and describe a different software development process model

General Background Preparation

- If you don't have it yet: Install and learn LaTeX— tug.org/texlive
- Read **RSD** chapters 1–4
- Review quantification, sets relations, functions, . . .
  (RSD 4; LADM 8,9,11,14; Using Z 3–8)
- Have a look at the C99 standard
- (If you don't have a UNIX-like system yet: Install **Linux** or ***BSD**)
- Install **Frama-C**
- (If you don't know functional programming yet:
  - Learn **Haskell** — haskell.org
  - or **OCaml** — ocaml.org

  and look in particular at simple tree datatypes)

Specify Finding the Maximal Element of an Array

```
int n;
int a[n];
```

$$\{ \quad ? \quad \} \; \mathsf{findMax}(n, a) \; \{ \quad ? \quad \}$$

Write pre- and post-conditions as predicate logic formulae!