Introduction To Formal Methods

OMSE 522 Lecture 1
April 5, 2001
Bart Massey
We'll get through as much of this as we can today, and finish the important parts of the rest tomorrow.
Part 1: Meta-Information

- What's with this course, anyhow?
- Role in OMSE
  - vs. past offerings
  - as background

This course serves two roles in OMSE:
+ The obvious: instruction in formal methods
+ Review of CS concepts: esp. discrete math + algorithms
About Me

- Bart Massey <bart@cs.pdx.edu>
  - algorithmic software development
  - programming language design and implementation

E-mail is always the best way to get in touch with me (Bart) quickly. If you want to talk on the phone, e-mail me a number to call you back.
About This Course

- Course format: Lectures/discussions + much out-of-class
  - reading
  - work
- Small class
  - attendance is important
  - must be prepared for class

The reading for the first week is the most brutal, but after that there will be homeworks, quizzes, etc. for a while. This course should be do-able, but it may not be easy.

Trying to keep discussion going in a small class like this is tough. You don't want me standing and reading you slides for 3 hours, so try to be ready to jump in whenever you need more information, can supply more information, or just have a comment on the material.
Administrative

- Parking: get permit, use SW lot
- Academic Honesty
- Videotapes + Web:
  - [http://www.omse.org/omse522](http://www.omse.org/omse522)
  - Username: omse522 Password: ?
- E-mail:
  - Instructor <bart@cs.pdx.edu>
  - Everybody <omse522-discuss@cs.pdx.edu>

I will give you the password in class. I'm not really picky about this, just want to protect some stuff from freeloaders and webcrawlers... :-) 

We'll help you get set up with the mailing list RSN. I use e-mail *heavily* in these courses, so we'll want to get your e-mail ASAP. The first assignment requires you to send me e-mail: make sure there’s a good address in it.
Course Support

- Office hour(s)?
- Help sessions?
- E-mail!
- Tools, accounts, etc.

Last time this course was taught, we got by without Saturday remedial help sessions. This time around, if it is necessary, let me know, and I'll set it up. I want to help you.

Z is much easier to work with given tools. There's a lot of them out there, but mostly for UNIX environments. I'll give you all accounts on my work UNIX box, but many of the tools are graphical. We'll try to figure this out as we go along: fortunately it'll be a while before you care.
Quizzes, Homework, Etc.

- Why?
  - theory/formal systems cannot be learned in class
  - keep everyone honest
- This is not a CS course
  - straightforward problems/questions
  - will keep practice in mind

Those of you remembering your discrete math homework and quizzes should be pleasantly surprised. We’re not trying to get you to stretch your brain around tough concepts, just to learn some unfamiliar notation for simple ideas. If approached in the right spirit, this will be fun...
Student Presentations

- Why?
  - demonstrate understanding
  - substitute for final
- What:
  - 1-1.5 hours in weeks 8, 9
  - talk w. slides

We won't nail this down until closer to the end, but thought I should give a heads-up now.
Course Texts

- Jonathan Jacky: *The Way Of Z*
  - obtain *this week* (total-info.com)
  - practical, ground-up description
- (Hinchey & Bowen: *Applications of Formal Methods -- OOP*)
  - selection of case studies
  - should convince you this stuff is for real

Jacky book is very popular. Jacky works with the radiation oncology dept. at UW, and was instrumental in understanding the Therac-22 situation. There's a link on the web site to his page about the text, including errata.

The Hinchey & Bowen book is just a collection of papers, but they are well-chosen, interesting, and to the point of this course. I’m going to try to get a copy of the later version of this collection if I can. (The one I really wanted to use was $120, and I decided to spare you all that. :-}
Other Z Sources

- Michael Spivey: *The Z Notation: A Reference Manual*
  - the definitive Z book (ANSI/ISO nws.)
  - good read
  - OOP, but available online!
- J.B. Wordsworth: *Software Development With Z*
  - OOP, hard to find
  - was first choice for course
- Web/online stuff

We can have hardcopy of the Spivey book printed for the usual small charge. Look at it and let us know if you want one.

Follow the links on the course web page: they're fairly nice.
Course Roadmap

- This week: Intro. to Formal Methods
- Next few weeks: math, logic, Z
  - try to emphasize relevance
  - toughest part of the course
- Couple of weeks: toy case studies
- Couple of weeks: real case studies (you)
- Final week: Status of Formal Methods

There will probably be a couple of guest lectures.
Z

- British, hence "Zed"!
- Standardized notation for math & logic plus other stuff
- Chosen for this course due to
  - simplicity
  - popularity
  - versatility

We'll be seeing a lot of it, so I wanted to make it clear to everyone what's going on.
Part 2: Formal Methods

- What is a "formal method" in SE anyhow?
  - *not* just a process or policy mechanism (e.g. CCBs)
  - *not* just a rigorous protocol (e.g. formal inspections)
  - use of a symbol system to model or analyze a software system

The term "formal methods" is a bit of a misnomer, but it'll do until something better comes along.
Symbol Systems

- Components
  - symbols
  - symbol manipulation rules
  - mappings between symbols and real world
- Examples
  - linear algebra
  - boolean algebra
  - programming language
  - baseball scoreboard

Some of the pain of mathematical methods may be relieved by thinking of them as formal notations akin to programming languages. This may also make it more obvious why we care about them.

``Baseball scoreboard?'' Sure. You put up symbols (numbers representing various states of the game) and manipulate them in well-understood ways to provide a model of a baseball game useful for analysis. This stuff doesn't have to be fancy/technical. (C.f. ``scoreboarding'' in computer architecture.)
Modeling and Analysis

- Modeling: mapping complex real system to simpler symbol system
- Analysis: mapping results of symbol manipulation to properties of real system

I'm using the word "analysis" a little differently than some might, but I think it's fair. The basic idea is modeling=abstraction, analysis=concretization. Note that analysis as decomposition is similar to analysis as concretization...
Doing Things The Hard Way

- Is “Just Do It” the hard way?
  - consequences of failure may be high
  - real behavior may be unobservable
  - measurement may be tough
- Are formal methods the hard way?
  - “Math is hard!”
  - modeling and analysis arrows are hard
  - eventually have to just do it anyhow

``Just Do It" ...Nike

``Math is hard!" Mattel's "Talking Barbie"

The former set of difficulties tends to be underestimated in practice, and the latter overestimated.
Extremely Brief History

- Formal SE methods precede computers (!)
  - λ calculus
  - Turing machine
- Most early SW systems built without
- Reinvented in 60s in response to high defect rates (Mills et. al.)
- Still not widely deployed

The tension between CS and SE is perhaps nowhere better reflected than in the attitudes toward formal methods. Unfortunately, there is often more heat than light here.
The first is the sort of obvious/traditional/accepted domain of formal methods.

The second works surprisingly well, but often is institutionally difficult. (Why is the SW complex or poorly understood? Often because of a busted development organization. These are hard to sell formal methods to.)

I'll expand on the third momentarily.
Formal Methods and Modularity

- Many formal descriptions essentially of ADTs
- ADTs with formal descriptions are trivial to reuse
  - in principle
  - in practice

This is perhaps one of the least-appreciated reuse tricks around. Why? Because it's not a silver bullet (no surprise): it's Superman vs. the Chinese Army again.
Correctness Proofs Don't

- TANSTAAFL: “I have only proven this correct, I have not tried it.”
- Can be mistakes in “proofs”!
  - problem worse than it sounds
- Can miss critical details
  - during modeling
  - during analysis

TANSTAAFL=``There Ain't No Such Thing As A Free Lunch" ...Heinlein

``I have only proven..." ...Donald Knuth

Avoid mistakes in your proof? Prove the proof correct. Yeah, right. Besides, Godel says you can't win anyhow.

Abstracting critical detail, or mis-concretizing, is quite common. This is one of the reasons for FM+test+inspect.
Formal Methods Are Expensive

- Mathematician/SEs are expensive!
- Learning curve is steep
- Tools are necessarily weak
- But *systems* requiring formal methods are hard

``Tools are necessarily weak": powerful systems imply difficult automation. E.g. blocks-world planning.

Sometimes the cost of formal methods are overestimated, since they are often used to tackle problems that would have been expensive no matter how they were solved.
There Is An Upside

- Formal methods *complement* traditional methods
- Formal reasoning mindset helps with traditional methods
- Some projects require formal methods
- The “*quality pays' thesis*” may still apply

More on the first point later.

The difference between a thesis and a theorem is important.
Formal Methods vs. Lifecycle

- Requirements specification +++ (Z, B)
- Design ++ (B, VDM)
- Implementation ++ (B-method, analyzers)
- Testing +++

Pluses indicate where it seems to help the most. The parenthesized systems are just a few examples.
Requirements Specification and Formal Methods

- Why requirements specification?
  - common source of defects
  - used for validation
  - Boehm's rule; leverage
- What to do
  - write down formal model
  - validate model
  - formal model + corrected informal spec.
    = good requirements spec.

Requirements errors are common and the most expensive kind (as you know).
What Is In A Requirements Spec?

- Domain constraints: modeled via sets, graphs, grammars
- Functional constraints: modeled via functions, relations
- Behavioral constraints: modeled via modes = states using statecharts, FSMs, etc.
- Others: e.g. performance, quality

These last things are more problematic to address using FMs.
Why Not Just “Do Math”?

- Physics, other eng. disciplines: ad-hoc applications of mathematics
- SW different: “fluid reality”; requires
  - general notation
  - mutable / customizable
  - strong formal semantics
- Z etc. meets this need

Note that I'm fudging a bit here: physics and atom engineering really do use various formal methods, e.g. FEM.
Formal Methods In Design

- Build on FM in requirements spec.
- Iteratively refine formal description
- End result: code with known properties
  - C.f. given code, produce correctness proof

The keyword here is to refine.

Note that if you are simply handed code, proving it correct is a nightmare. It's much easier to build correct code, and use the proofs as just checks.
Formal Methods In Implementation

- Note that PLs have automated FM:
  - syntax/grammar checker
  - typechecker
  - static analyses e.g. Java undef., C lint
- Good, but too late
  - open research area: leverage FM in design, analysis for implementation

We're weird about formal methods in programming languages. They must be:

+ exceptionally simple to understand
+ completely automated in analysis
+ traditionally accepted

or they don't get adopted.
FM In Testing: Static Info.

- FM requirements spec. approximates “Holy Grail”: effective oracle
- Abstraction provides well-founded partition of test spaces

The key is of course the reversal of the magenta arrow. The label there could be ``modeling'' as well...
FM In Testing: Dynamic Info.

- Known invariants
  - complete, correct assertions
- Known behavior
  - complete coverage (e.g. FSMs)
- Meaningful failures
  - more likely to fail vs. wrong answer
  - easier to diagnose failures

What a strong type system (e.g. Java) does for testing, strong formal methods do on steroids.
FM vs. Testing, Inspection

- Testing complements inspection
- FM complements both
  - leverages testing
  - cannot test for incomplete requirements
  - exposes deep semantic defects;
    inspection better at surface defects
  - more tool opportunities than inspection

Testing + inspection is very good, but FM + testing + inspection is much better, which is why it is mandatory in most safety-critical systems.
FM In Maintenance

- Clear, complete, correct description at all design stages
- Can get help from tools in propagating changes
- SW system may need less maintenance

Actually, there are a couple of reasons why there is not a huge amount of experience here. First, there's the third bullet above. Second, due to the high (perceived?) development cost of systems using FMs, organizations are reluctant to perturb them and potentially re-incur some of these costs.
“Lightweight Formal Methods”

- Can get some of the advantages described w/o going whole hog
- Carefully tailored method to solve a specific problem
- Still needs good notational and theoretical support

Hopefully, we can play with a little of this sort of thing in this course.
Our Formal Method: \textit{Z}

- \textit{Z} is \textbf{not} a programming language: “\textit{Z} notation”
- \textit{Z} is more than just discrete math + logic
  - but not much more (hence its use here)
- \textit{Z} is oriented toward requirements development
  (but can be used throughout)

Why a notation? Trade automatabilty for expressivity.
Part 3. A Simple Example

- “Birthday Book” from Spivey, 
  Z Reference Manual
- Referenced throughout Z literature
- No details / scary math yet: just overview

“Example” may be overstating the case. I just want to give a broad-brush sketch of how this stuff can go.
Birthday Book Requirements

- Provide a piece of software which
  - associates names with birthdays
  - issues birthday card reminders for requested date
- Usual problems of natural language spec.
  - invalid input?
  - unique birthday per name?
  - deleting names?
  - reminder methodology?

Are there other problems with the NL spec?
Z Requirements Model

- Model state of book as mathematical objects + logical invariants
- Model restrictions on input as logical predicates
- Model operations on book as state changes

What does the notation do? It makes these things easy(-ier)
Z Design and Implementation

- Design
  - Implement state as
    - two arrays
    - count
  - Model arrays + count mathematically
  - Provide formal mapping from req. to model

- Implementation
  - Code from design straightforwardly
  - Use assertions to enforce invariants

None of this is terribly hard, and you'll be doing it yourselves shortly.
How Z Helps

- Several requirements problems caught up front
  - incomplete input specification: dates
  - incomplete behavior specification: invalid requests
- Simple model yields simple, clear code
- Not difficult!
- Modular: can easily use as component

Studies of defect rates and types suggest that there's no such thing as an "easy problem" (Jacky’s point in his intro.). As always in SE, hard work up front makes it easy :-).
Final Comments

- Formal methods may seem awkward and unnatural at first
  - so did computer programming
- There is no need for magic here
  - do not need deep mathematical insight to use FM
  - but do need to use simple mathematical tools
- The final chapter is not yet written

Will formal methods take over or fade away? Unlikely either. But I suspect that as the distinction between hackers and users blurs, real SEs will be distinguished partly by their ability to use formal methods when appropriate.
Next Week

- "Review": everything you should used to have known about math, logic, etc.
- Reminders:
  - obtain texts
  - catch up readings
  - investigate web, mailing lists