COURSE INFORMATION

Text:

No text, many handouts (see below)

Class structure:

Each student will prepare two reports on two formal methods topics. (Topic suggestions are below.) Each class period, a student and the instructor will jointly cover one selected topic.

Evaluation:

Available grades:

non-completion: Incomplete, Withdraw, etc.

completion: A A- B+ B B- C

A:	reserved for superior performance
A- or B+:	expected grade for conscientious performance
B:	adequate work
B-:	barely adequate
C:	equivalent to failing

Grading Options:

1. Performance Quality: attendance, participation, assigned exercises
2. Grading Contract: specify a set of behaviors and an associated grade.
3. Self-determined: negotiate with instructor

Discussion:

If you prefer a clearly defined agenda, if you do well with concrete task assignments, or if you need a schedule of activities for motivation, then Option 1 is a good idea.

If you already understand the field, if you plan to excel in a particular area, or if you need clear performance goals for motivation, then Option 2 is a good idea.

If you are not concerned about grades, if you intend to do what you choose anyway, or if you are self-motivated, then Option 3 is a good idea.

I will notify any student who is not on a trajectory for personal success.

Course content:

Formal methods is a body of mathematically-based techniques, often supported by reasoning tools, that offers rigorous ways to model, design, and analyze systems.

We will explore a number of specific applications of formal methods. The course will focus on implementations of tools and techniques and the use of these tools. Each class, the instructor will give a lecture on the mathematical techniques of a particular formal method. During the same class period, students will present their research and experiences with the implementation of that technology.

Although the Computer Science community limits formal methods to applications of logic and predicate calculus, this course will take a slightly broader viewpoint. Numerical and algebraic techniques such as matrix algebra, probability theory, and integer theory will be excluded, but exotic symbolic approaches such as fractals, cellular automata, and boundary mathematics will be included as possible topics. Pure programming languages (Prolog, ML, Haskell, LISP, Mathematica) are also valid topics.

Individual homework will consist of a short selected reading on each topic, personal exploration of implementations of at least two formal tools, one or two class presentations, and whatever exercises necessary for understanding.

A Quote from the Oxford Group

"There's a battle going on in computer science that will probably never be fully resolved, between those who think programs are fundamentally mathematical, and those who eschew mathy techniques as being too tedious for use with real-world programs. Despite a layperson misperception to the contrary, most programmers avoid math just as most nonprogrammers do, with the result that more than 99% of software is developed today as nonmath.

Formal methods is the name for the techniques of mathematically proving that programs do what they're supposed to. The theory is that programs aren't physical objects, they are ideas; they don't break down, and they don't wear out, the way physical objects do. A perfect program will therefore remain perfect forever. Formal methods exist to make such perfect programs, compared to which even the most well-crafted nonmath program is fundamentally a buggy slapped-together sloppy mess.

It would be nice if formal methods were more widely accepted, because as programs grow larger and larger the interspersed bugs make them more and more unreliable. But formal methods slow the pace of program development so much, and fit so poorly into the messy but productive real world, that they are used only rarely even in potentially life-threatening systems.

Some Formal Techniques

The list of topics which follows is organized by mathematical techniques, with application areas following the mathematical topic (asterisks mark recommended topics).

Propositional calculus (Boolean logic)**
circuit design, hardware verification, Boolean minimization, control theory

Predicate Calculus**
expert systems, specification languages, theorem provers, correctness and verification

Logic Extensions
non-monotonic reasoning, temporal logic, process algebra

Mathematical Induction and Recursive Function Theory**
proof technique, recursive programming, programming

Relational Calculus*
relational databases, constraint solving

String Rewrite Theory*
mathematical computation, process modeling, parsers and compilers

Theory of Computation*
worst-case algorithms, time and space complexity

Fractals
computer graphics, compression, computer art

Binary Decision Diagrams
hardware modeling

Lambda Calculus and Combinators
functional programming

Group Theory and Modern Algebra
coding theory, 3D motion

Finite State Automata
state space problem solving, string recognition, state transition systems

Cellular Automata
chaos modeling

Boundary Mathematics
visual languages, logic and numerical simplification, parallel processing

General Systems Theory
systems modeling, control theory

References

General:

Bavel (1982), Math Companion for Computer Science, Prentice-Hall
Gilbert (1976), Modern Algebra with Applications, Wiley
Grassmann and Tremblay (1996), Logic and Discrete Mathematics, Prentice-Hall
Gries and Schneider (1993), A Logical Approach to Discrete Math, Springer-Verlag
Grimaldi (1999), Discrete and Combinatorial Mathematics, Fourth edition, Addison-Wesley
Lucas (1985), Introduction to Abstract Mathematics, Second edition, Ardsley House
Wolfram (1996), The Mathematica Book, Third edition, Cambridge Press

Specific:

Aho, Sethi and Ullman (1986), Compilers, Addison-Wesley
Barwise and Etchemendy (1993), The Language of First-Order Logic, Third edition, CSLI Stanford
Forbus and DeKleer (1993), Building Problem Solvers, MIT Press
Genesereth and Nilsson (1987), Logical Foundations of Artificial Intelligence, Kauffman
Hopcroft and Ullman (1979), Introduction to Automata Theory, Languages, and Computation, Addison-Wesley
Lakatos (1976), Proofs and Refutations, Cambridge U. Press
MacLennan (1990), Functional Programming, Practice and Theory, Addison-Wesley
Manna and Waldinger (1985), The Logical Basis for Computer Programming, Addison-Wesley
Plasmeijer and vanEekelen (1993), Functional Programming and Parallel Graph Rewriting, Addison-Wesley
Wos, Overbeek, Lusk and Boyle (1992), Automated Reasoning, Second edition, McGraw-Hill

Web Pointers

Oxford University Computing Laboratory
http://www.comlab.ox.ac.uk/archive/formal-methods.html

BYU Laboratory for Applied Logic
http://lal.cs.byu.edu/

NASA Langley Research Center Formal Methods Program
http://shemesh.larc.nasa.gov/fm.html

Swedish Institute of Computer Science
http://www.sics.se/fdt/research97.html

UC Davis Programming Languages and Verification Laboratory
http://avalon.cs.ucdavis.edu/

Stanford U. Center for Formal Methods
http://www-formal.stanford.edu/jmc/math.html

Warsaw U. Applied Logic Group
http://zls.mimuw.edu.pl/english.html

UC Berkeley Design Technology Warehouse
http://www-cad.eecs.berkeley.edu/

A Computational Logic
http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html

Formal Methods in Software Engineering
http://wwwsel.iit.nrc.ca/projects/fm/fm.html

Formal methods around the world
http://lal.cs.byu.edu/other_FM.html

Software Development using Formal Methods Syllabus
http://www.mcs.salford.ac.uk/sdformal.html

Bibliography on software engineering and formal methods
http://bavi.unice.fr/Biblio/SE/Contrib.html

Seven Myths of Formal Methods
http://www.progsoc.uts.edu.au/~geldridg/frsd/ass1/7myths.htm

Formal Methods - selected historical references
http://docs.dcs.napier.ac.uk/DOCS/GET/jones92a/document.html

Books
http://www.rspa.com/spi/formal.html

Rough Syllabus

NOTE: TOPICS may change by class consensus.

Class meeting	Topic
1)	introduction
2)	overview of formal methods
3)	complexity, proof techniques
4)	proof systems, boundary logic
5)	Boolean minimization, bdds
6)	abstract domains
7)	pattern-matching and unification
8)	recursive function theory
9)	lambda calculus
10)	combinators
11)	theorem provers
12)	theorem and program proving
13)	Mathematica, string rewrite
14)	relational algebra
15)	finite state automata
16)	cellular automata
17)	abstract algebra and group theory
18)	fractals
19)	to be determined
20)	review and summary