CS294-2 Software Synthesis

Spring 2006

Ras Bodik

office hours: Tue 11-12, Thu 3-4

Outline

- nature of this cs294
- software synthesis
  - what it is and why study it now
- my interests in synthesis
  - sample of what problems it can solve
- course format
- course agenda
- overview of papers
  - how broad is synthesis in this course?
- student introductions
- plan for coming lectures

Nature of this CS294

- **starter for a long-term research project**
- **Berkeley tradition:** OSQ, ROC, IRAM, RISC
- **purpose:**
  - bring together students with diverse backgrounds
    - not all to work in the project after cs294
  - learn the field:
    - prior work, failures, successes
  - set a research agenda
    - refine it with cool course projects
- "grad school experience in one semester"

What is software synthesis?

- Sibling to verification
  - **verification:** write code; verify it meets the spec
  - **synthesis:** generate code that meets the spec
  - so, we get both correctness and productivity
  - my bias towards productivity (neglected)
- Synthesis techniques: a diverse spectrum
  - from fancy compiler optimizations, to deductive synthesis, to genetic programming
- Often a form of search is involved
  - "when in doubt, use brute force," Ken Thomson

Why synthesis now?

- Need:
  - software development pain became unbearable
  - productivity, correctness > performance
  - everybody is a programmer
  - parallel machines everywhere, how to program them?
- Opportunity:
  - verification technology maturing
    - one view: synthesis = verification + search
  - software process becoming more formal
    - programmers more willing to write specs
  - moore's law allows search
  - parallelizable search
  - domains bigger → DSL more economical
  - DSL: declarative, models → easier to synthesize

How I got interested in synthesis

Problem 1: Scientific computing

- parallel programs written from scratch
- minor change to algorithm causes major rewrite
- automatic parallelization not a success in practice

Vivek Sarkar posed a problem:

"How could a domain expert and the parallel hacker collaborate? Two roles, two aspects?"

- **domain expert** (bio, crypto, nukular): designs the algo
- **hacker:** knows caches, vectors, communication

How I got interested in synthesis

Problem 2: Object-oriented API programming

- API's are useful, indispensable …
- … but have 10,000s of methods
- learning curve, like climbing a glass wall

- Doug Kimelman, Mark Wegman asked:
  "Can you mine examples of API usage and index them?"
- My alternative view, more ambitious:
  "synthesize desired code in response to a query"

How I got interested in synthesis

Summary:

- my interest in synthesis problem-driven
- only in part an evolution of past interests, expertise
- means: I know nothing about synthesis
- no better way to learn than in cs294

Course format

- Read papers and discuss them
  - discuss, or ideally brainstorm
  - for that, we need to a set of challenge problems
  - more on this later
- To prepare for class
  - read the paper and email me a brief summary
  - summary: may include provocative questions, etc

Course format

- Each student will present one paper
  - rather than preparing lecture write-ups
  - if you want, think of it as leading a discussion
    - with powerpoint
  - I will discuss with you lecture outline beforehand
- Each week, one student presentation
  - Some guest lectures, too (TBD)

Course format

- Projects: the usual
  - literature review
  - algorithm design, proofs
  - implementation
  - or all of the above
- Class presentation + written report

Course research agenda

- What do we want to learn about the papers?
  - good to have a problem in mind that we hope these problems will solve
- Some of my favorite problems: how to develop …
  - general-purpose synthesis,
  - … embeddable in Java, FORTRAN
  - … teachable in CS4
- Your favorite problems here (dreaming allowed):
  -
  -

Topics

- Archeology
  - Old fun classics
  - Questions: why not in Eclipse, Visual Studio by now?
- Successful (working) systems
  - What did they do right?
  - Can be adopted to other problems?
- New problems
  - potentially solvable with synthesis
- New technologies
  - Scalable solvers, modern theorem provers
- New mindsets
  - Semi-automatic is good enough, or better

Overview of papers (initial list)

- Deductive software synthesis:
  - prove that desired program exists
  - the (constructive) proof is the program
  - often, counterexample is the proof
- Papers:
  - "Toward automatic program synthesis"
    - 1971
  - "KIDS: A Semi-Automatic Program Development System"
  - Amphion (NASA)
    - Two real systems

Overview of papers

- Transformational synthesis
  - ex.: transform recursion into iteration (Fibonacci)
- Papers:
  - "A Transformation System for Developing Recursive Programs", 1977
  - "Program improvement by internal specialization", '81
  - Synthesis of concurrent garbage collectors (guest)
    - prove GC correct by instantiation from a simple one

Overview of papers

- Program differentiation
  - "Finite Differencing of Computable Expressions" '82
  - "Incrementalization across object abstraction" '05
    - write OO containers in specification style
    - ex.: hashtable.size() iterates and counts the elements
    - then automatically incrementalize to an efficient version
    - hashtable maintains a _size field, updated by insert(), …

Overview of papers

- Superoptimizers:
  - a search for the best assembly code sequence
- Papers
  - "Superoptimizer: a look at the smallest program"
    - enumerate and test for correctness
    - supplemental reading
  - "Denali: a goal-directed superoptimizer"
    - derive and schedule optimally
  - demo: Aha (documentation), code

Overview of papers

- Programming by demonstration, scenarios:
  - Learning Shell Scripts with Version Spaces
  - Come, Let's play
    - reactive systems
    - ask Ras for the book
  - Watch What I do
    - a list of interesting papers
    - book online

Overview of papers

- Synthesis with partial programs:
  - programs with holes (templates)
- Papers
  - Program synthesis as machine learning: ALisp
    - programs implement agents
    - holes synthesized via learning
  - Programming by Sketching
    - high-performance kernels
    - holes synthesized s.t. program behaves like the spec

Overview of papers

- Scientific computing
  - Synthesis of irregular codes
    - 50 or so sparse matrix representations
    - how do you generate the code for them?
  - FLAME: Formal Linear Algebra Methods Environment
  - sparse code synthesis (Yelick et al)
- Schema-based synthesis:
  - AutoBayes

Overview of papers

- Object-oriented programming, components:
  - A pragmatic approach to software synthesis
  - Prospector
  - Application generators
- Genetic programming:
  - paper TBD

Student introductions

- name, research project and advisor
- why are you taking or auditing the course?
- a problem in your work that synthesis may solve?
  - security,
  - bebop (sparse matrix codes)
  - CAD simulation codes, fault-free code, parallel embedded
    - multimedia, automotive
  - visualization, high-level languages, usability, DSLs
  - distributed data structures
  - refactoring
  - verification
  - ALisp

Next two weeks

- Lectures: establish some challenge problems
  - Thu: Sketching (Ras)
  - Tue: Prospector (Ras)
  - Thu: TBD,
    - maybe brainstorm on course projects
    - maybe talk about research agenda

First homework

- Homework for Thu:
  - no reading, but go over papers listed on the web site
  - understand the scope of synthesis as defined in cs294
  - **task:** find another paper we may want to civer
    - ok to broaden the scope of what we mean by synthesis
    - no need to be interested in presenting the paper yourself
  - **email me:**
    - the link to the paper
    - justification why this paper concerns software synthesis
  - I will add some of the papers to the list

Sign up for paper presentations

- By Tue next week:
  - sign up for a paper,
  - more details later