Expeditions in Computer Augmented Program Engineering

http://excape.cis.upenn.edu/

Cornell, Maryland, Michigan, MIT, Penn, Rice, UC Berkeley, UCLA, UIUC

NSF Expeditions PI Meeting, May 2013
Software Design Methodology

- What has changed:
  - Programming languages
  - Libraries
  - Verification technology

- What has not changed:
  - Programming is done by experts
  - Fully specified by conventional programming
  - Verification phase is distinct from design

Can we leverage modern analysis tools and increased computing power to revolutionize the task of programming?
Synthesis: A Plausible Solution?

- **Classical:** Mapping a high-level (e.g. logical) specification to an executable implementation
  - Theoretical foundations: Church (1960s)
  - Derivation of programs from constructive proofs (e.g. Kestrel)
  - Synthesis from temporal logic specs: Clarke/Emerson (1980s)
  - Refinement in model-based design
  - Ongoing progress, but many challenges remain...

- **Recent shift in focus:** Integrating different styles of specifications in a consistent executable
Sketch: Program completion

Err = 0.0;
for(t = 0; t<T; t+=dT){
  if(stage==STRAIGHT)
    if(t > ??) stage= INTURN;
  }
  if(stage==INTURN)
    car.ang = car.ang - ??;
    if(t > ??) stage= OUTTURN;
  }
  if(stage==OUTTURN)
    car.ang = car.ang + ??;
    if(t > ??) break;
}
simulate_car(car);
Err += check_collision(car);
}
Err += check_destination(car);

When to start turning?
Backup straight
How much to turn?
Straighten

Enables programmers to focus on high-level solution strategy
ExCAPE Vision

Harnessing computation to transform programming:
Programming made easier, faster, cheaper
Synthesis Tool: Intelligent Assistance

- Designer expresses “what”, possibly using multiple input formats
- Synthesizer discovers new artifacts via integration and completion
- Synthesizer solves computationally demanding problems using advanced analysis tools
- Interactive iterative design
- Integrated formal verification
Research Organization

- **Tools and Evaluation**
- **Design Methodology**
- **Computational Engines**
- **Challenge Problems**
  - Apps for Mobile Platforms
  - Multicore Protocols
  - Networked Systems
  - Robotic Systems
- **Education and Knowledge Transfer**
Talk Outline

- Design tool for distributed protocols
- Synthesis for programming robots
- Synthesis to support online education
- Summary of ExCAPE activities
Goal: Simplify Protocol Design

- Design challenging due to asynchronous model of communication
- Cache coherence protocols, Distributed coordination algorithms
- Successful application domain for formal verification / model checking
- Correctness involves both safety and liveness properties
- Proposed solution: Allow programmers flexibility

Protocol = Skeleton based on Extended-Finite-State-Machines
+ High-level requirements
+ Example behaviors
Computational Problem

- **Inputs:**
  - Variable types and corresponding expression grammar
  - For each process,
    1. Control states of EFSM
    2. List of all variables, input/output messages
    3. Set of concrete examples + symbolic constraints
  - High-level requirements (invariants and temporal logic formulas)

- **Solution strategy:**
  - **Expression Inference:** For each EFSM transition, generate expressions for guards and updates. Solution uses Counter-example-guided-inductive synthesis using SMT solver Z3
  - Check if resulting protocol meets all requirements, using a model checker (Murphi) and if not, report a counter-example
Challenging Case Study: SGI Origin protocol

- **Source:** Laudon and Lenoski; The SGI Origin: A CCNUMA highly scalable server; ISCA 1997

- Directory-based MESI protocol that handles multiple concurrent requests to same requests over unordered network

- Textual description directly leads to protocol skeleton, and symbolic (incomplete) descriptions of most of the transitions

- During debugging, programmer focuses on local fixes of counterexamples and adds concrete examples

- Final iteration required 30 min synthesis time (with 5 Million states explored by Murphi)

- SMT solver / model checker in the loop is feasible for programming
Synthesis for Robot Programming

Goal: Allow end-users to program robotic behaviors

Automatically
(Provably Correct)
Robotic controllers: Research Challenges

- How to consistently integrate physical constraints, sample trajectories, safety rules, and language/temporal-logic requirements?

- How to explain infeasible requirements? How to suggest potential fixes?

- How to program a synthesis engine with completion strategies that take into account the physical and continuous nature of robotics (power, safety, environment traversability)?

- How to address optimality and performance?

- How to evaluate human-robot interaction?

- How to generate control that ports across different robots (different dynamics, control capabilities, safety considerations)?
LTLMoP: Robot control from structured English
Research Results

- Improving the scalability of core engine for mapping Temporal Logic formulas to Controllers:
  - Synthesis with identifiers (Kress-Gazit, Seshia)

- Synthesis of cost-optimal plans (Kress-Gazit)

- Motion planning in partially unknown environments (Kavraki, Vardi)

- Synthesis of controllers with robust performance in presence of uncertainties
  - Theory of robustness for hybrid systems (Tabuada)

- Accuracy in mapping discrete actions to continuous-time trajectories with durations (Kress-Gazit)

- Automatic generation of environment assumptions (Alur, Topcu)
Ongoing Case Study: Robotic Waiter

- Challenges: Scalability (items, costumers), uncertainty in sensing and actuation, optimality of behavior, fault recovery
- Future plans: exploit symmetries, robust synthesis, task specific abstractions

Initial demo using LTLMoP (Kress-Gazit) and OMPL (Kavraki)
Synthesis for Online Education

- **Emerging opportunity: MOOCs**

- **Challenge: Personalized feedback on assignments**
  - Manual feedback by TAs (not scalable)
  - Grading by peers (not reliable)
  - Evaluation on test cases (how to translate failed tests to errors?)

- **Application for ExCAPE tools for synthesis**
  - Introductory programming assignments (Solar-Lezama, MIT)
  - Scheduling problems in Embedded Systems course (Seshia, UC Berkeley)
  - DFA construction in Theory of Computation (with Hartmann, UC Berkeley)

  see automatatutor.com
```python
def computeDeriv(poly):
    result = []
    for i in range(len(poly)):
        result += [i * poly[i]]
    if len(poly) == 1:
        return result
    else:
        return result[1:]  # remove the leading 0
```

Sample Problem: Derivative of a Polynomial
def computeDeriv(poly):
    deriv = []
    zero = 0
    if (len(poly) == 1):
        return deriv
    for e in range(0, len(poly)):
        if (poly[e] == 0):
            zero += 1
        else:
            deriv.append(poly[e] * e)
    return deriv

The program requires 3 changes:

- In the return statement `return deriv` in line 5, replace `deriv` by `[0].`
- In the comparison expression `(poly[e] == 0)` in line 7, change `(poly[e] == 0)` to False.
- In the expression `range(0, len(poly))` in line 6, replace 0 by 1.
Computational Engine: Sketch Synthesis Tool

S: Student Solution

All Programs

Programs satisfying spec

Minimum changes

4 changes

2 changes

1 change
## Autograder Experiments (MIT 6.00)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>TestSet</th>
<th>Feedback</th>
<th>Percentage</th>
<th>AvgTime(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>prodBySum-6.00</td>
<td>268</td>
<td>218</td>
<td>81.34%</td>
<td>2.49</td>
</tr>
<tr>
<td>oddTuples-6.00</td>
<td>344</td>
<td>185</td>
<td>53.78%</td>
<td>2.65</td>
</tr>
<tr>
<td>compDeriv-6.00</td>
<td>103</td>
<td>88</td>
<td>85.44%</td>
<td>12.95</td>
</tr>
<tr>
<td>evalPoly-6.00</td>
<td>13</td>
<td>6</td>
<td>46.15%</td>
<td>3.35</td>
</tr>
<tr>
<td>compBal-stdin-6.00</td>
<td>52</td>
<td>17</td>
<td>32.69%</td>
<td>29.57</td>
</tr>
<tr>
<td>compDeriv-6.00x</td>
<td>918</td>
<td>753</td>
<td>82.03%</td>
<td>12.42</td>
</tr>
<tr>
<td>evalPoly-6.00x</td>
<td>541</td>
<td>167</td>
<td>30.87%</td>
<td>4.78</td>
</tr>
<tr>
<td>oddTuples-6.00x</td>
<td>1756</td>
<td>860</td>
<td>48.97%</td>
<td>4.14</td>
</tr>
<tr>
<td>iterPower-6.00x</td>
<td>2875</td>
<td>1693</td>
<td>58.89%</td>
<td>3.58</td>
</tr>
<tr>
<td>recurPower-6.00x</td>
<td>2938</td>
<td>2271</td>
<td>77.30%</td>
<td>10.59</td>
</tr>
<tr>
<td>iterGCD-6.00x</td>
<td>2988</td>
<td>2052</td>
<td>68.67%</td>
<td>17.13</td>
</tr>
<tr>
<td>hangman1-6.00x</td>
<td>351</td>
<td>171</td>
<td>48.72%</td>
<td>9.08</td>
</tr>
<tr>
<td>hangman2-6.00x</td>
<td>218</td>
<td>98</td>
<td>44.95%</td>
<td>22.09</td>
</tr>
</tbody>
</table>

64% 9.9s
Rosette: Framework for developing synthesis-enhanced DSLs (Bodik)
Enhancements to Sketch to support modularity (Solar-Lezama)
Bridging the gap between reactive synthesis and supervisory control (Lafortune, Tripakis, Vardi)
Verified LLVM Infrastructure (Martin, Zdancewic)
Platform-based design for software synthesis (Sangiovanni-Vincetelli)
Synthesis of logic for avoiding concurrency bugs (Lafortune)
Component-based synthesis for probabilistic systems (Vardi)
Theory of regular functions for quantitative analysis (Alur)
Automated cloud configuration using synthesis (Alur, Loo, Parthasarathy)
Route Shepherd for configuration of routing protocols (Loo)
Synthesis of control + scheduling for wireless control networks (Pappas)
Programming for mobile platforms (Foster, Solar-Lezama)
User studies for improving programming notations (Hartmann)
Based on input format for SMTLib 2

Problem: Given a formula $\phi$ in an SMT theory with an extra function symbol $f$, and context-free language $L$ for templates, find an expression $e$ in $L$ such that $\phi[f/e]$ is valid

Basis for synthesis competition (to be held at CAV 2014)
Education and Outreach

- ExCAPE Summer School: June 13–16, Berkeley; 125 registrants
  Tutorials: Reactive synthesis (Vardi)
    Constraint-based program synthesis (Bodik)
    Synthesis for cyber-physical systems (Tabuada)
  + Talks

- ExCAPE Webinar: Monthly talks on diverse topics

- Sponsored workshops
  SYNT (at CAV 2013, by Solar-Lezama)
  Synthesis for robotics (at RSS 2013, by Kavraki and Kress-Gazit)
  Special session on synthesis at ACC 2013 (by Lafortune)

- Graduate course at Berkeley: Program synthesis for everyone

- K-12 programs: CURIE @ Cornell
Rotating Postdoc Program

- Each ExCAPE postdoc has two mentors, at two different institutions

- Year 2012-13:
  - Ruediger Ehlers (Robotics)
    - Mentors: Kress-Gazit (Cornell), Seshia (UC Berkeley)

- For the upcoming year:
  - Xiaokang Qiu (UIUC), Apps for mobile platforms
    - Mentors: Foster (Maryland), Solar-Lezama (MIT)
  - Indranil Saha (UCLA), Robotics
    - Mentors: Pappas (Penn), Seshia (UC Berkeley)
  - Christos Stergiou (UC Berkeley), Multicore protocols
    - Mentors: Martin (Penn), Tripakis (UC Berkeley)
  - Damian Zufferey (IST Austria), Networked systems
    - Mentors: Loo (Penn), Parthasarathy (UIUC)
Paradigm shift in synthesis:
Old: Allow more concise, high-level description
New: Designer uses multiple, natural formats,
Synthesis tool assists in discovering tricky logic

Paradigm shift in design tools:
Old: Any compiler transformation must be polynomial-time
New: Computational intractability not a show-stopper

Common theme: Guided search in a space of programs to find one that meets multiple design goals
A bit like model checking, but can be interactive!