ExCAPE
Expeditions in Computer Augmented Program Engineering

Rajeev Alur
University of Pennsylvania

Kick-off Meeting, June 2012
Programming Technology

Libraries → Program → Compiler → Executable

- **Libraries**: High-level programming abstractions (object-oriented, declarative, domain-specific..)
- **Program**: Semantics-preserving transformations (low-level optimizations, type inference ..)
- **Compiler**: 
- **Executable**: Platform
Software Design

Libraries → Program → Specifications → Tests

Compiler → Analysis Tool

Executable → Platform
Verification Technology

Program

Specifications

Tests

Automated verification

(model checking, static analysis, specification-based testing ..)

Analysis Tool

Executable

Platform
Challenges

- Software development still remains expensive and error-prone...

- What it means to “code” hasn’t changed...

- Verification/testing done after design
  - Costly system design cycle
  - Many reported bugs not fixed

- Computing power is transforming many engineering disciplines with the notable exception of programming itself
Opportunities

- Enormous computing power available on desktops of today’s programmers

- Impressive strides in formal verification technology
  - Highly optimized SAT solvers that can solve real-world problems
  - Off-the-shelf tools for static analysis, machine learning...

- Demand for new software development approaches
  - Receptive industry
  - Shifting goal of system design from performance to predictability
Synthesis: A Plausible Solution?

- Classical: Mapping a high-level (e.g. logical) specification to an executable implementation, impressive progress, but...
  - Is writing logical specifications easier, more natural?
  - Programmer still needs to figure out the entire logic
  - Isn’t this computationally infeasible for practical impact?

- Recent shift in focus: Integrating different styles of specifications in a consistent executable

  To be illustrated by 3 sample projects
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?
Turn
Straighten

Enables programmers to focus on high-level solution strategy
QuickCode: Programming by Examples

Ref: Gulwani (POPL 2011)

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(425)-706-7709</td>
<td>425-706-7709</td>
</tr>
<tr>
<td>510.220.5586</td>
<td>510-220-5586</td>
</tr>
<tr>
<td>1 425 235 7654</td>
<td>425-235-7654</td>
</tr>
<tr>
<td>425 745-8139</td>
<td>425-745-8139</td>
</tr>
</tbody>
</table>

- Infers desired Excel macro program
- Iterative: user gives examples and corrections
- Being incorporated in next version of Microsoft Excel

Enables non-programmers to program interactively
Paraglide: From Sequential to Parallel Code

Ref: Vechev et al (POPL 2010)

Sequential Program

```c
bool add(int key)
{
  atomic
  Entry *pred,*curr,*entry
  locate (pred,curr,key);
  k = (curr->key == key)
  if (k) return false
  entry = new Entry()
  entry->next = curr
  pred->next = entry
  return true
}
```

Minimal Synchronization

```c
bool add(int key) {
  Entry *pred,*curr,*entry
  restart:
  locate(pred,curr,key)
  k = (curr->key == key)
  if (k) return false
  entry = new Entry()
  entry->next = curr
  val= CAS(&pred->next,<curr,0>,<entry,0>)
  if (!val) goto restart
  return true
}
```

Architecture Description

- Target: Highly concurrent work queue in C/C++
- Infers minimal number of fences needed for synchronization
- Unexpected, correct, minimal solutions now deployed in IBM

Enables programmers to meet new programming challenges
Computer Augmented Program Engineering

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
Challenge Problems

- Representative of complexity: cyber-physical systems on networked, multi-core platforms
- Concrete design problems to guide tools and methodology
- Multiple challenge problems to avoid domain-specific solutions
Proposed Research

In each challenge area,

- Identify a concrete design problem for which new solutions can enable new applications
- Identify most promising synthesis-based solution strategies
- Develop theoretical foundations and algorithmic advances
- Build tools and prototypes
- Evaluate tools for scalability, user interaction, and programmer productivity
- Refine and advance computational/methodological solutions and tools

Cross-fertilize ideas and tools across challenge problems
Protocol Design from Concolic Snippets
Research Expertise

- Programming languages: Bodik, Foster, Solar-Lezama, Zdancewic
  Program analysis, programming environments, type systems, concurrency
- Computer-aided verification: Alur, Madhusudan, Seshia, Vardi
  Model checking, formal specification and verification, constraint solvers
- Control theory: Lafortune, Pappas, Tabuada
  Robust control, networked control, hybrid systems
- Embedded software: Alur, Sangiovanni-Vincentelli, Tripakis
  Model-based design, compositionality, portability
- Computer architecture: Martin
  Multi-core protocols
- Networked systems: Loo, Pappas
  Declarative networking, routing protocols, wireless networks
- Robotics: Kavraki, Kress-Gazit
  High-level design and implementation of robotic controllers
- Human-computer interaction: Hartmann
  Design and evaluation of user interfaces
Formal Methods: From Theory to Practice

- **Logic synthesis**
  - Key advances in logic minimization, placement and routing algorithms (1980s)
  - Today: EDA Synthesis industry is $3.5B

- **Hardware verification and model checking**
  - Key advances in symbolic model checking and SAT solvers (1990s)
  - Today: Routinely used in IBM, Intel, ... (within IBM, impact worth $0.5B)

- **Program analysis**
  - Foundations in abstract interpretation and dataflow analysis (1980s)
  - Today: ASTREE tool used within avionics industry

- **Software verification**
  - Advances in abstraction, refinement, and constraint solvers (2000s)
  - Today: Static Device Verifier part of Windows development
Impacting Industrial Practice

- Keys to transitioning academic research to industrial practice
  1. Market pull and industrial interest
  2. Algorithmic advances and computational tools
  3. Methodology for integration in design cycle

- Our plan: Advance computational tools and methodology, and demonstrate benefits on meaningful case studies

- Collaborators:
  Chitta (Willow Garage), Deshmukh (Toyota), Gulwani (Microsoft)

- Advisory Board:
  Field (Google), Fix (Intel), Godbole/Khurana (Honeywell), Gupta (NEC), Kuehlmann (Coverity), Lee/Godefroid (Microsoft), Mosterman (Mathworks), Wegman (IBM), Zave (AT&T)
Education and Outreach

- Annual workshop
  
  Academic and industrial participants

- Summer school
  
  Integrative and multi-disciplinary training

- Synthesis competition
  
  Benchmarks and tool evaluation

- Undergraduate education
  
  Course modules for CS and CE courses

- Attracting high-school students to CS and Engineering
  
  - Programming is not equal to coding
  
  - Projects in robotics
  
  - Collaboration with existing high-school programs at PI institutions
public class Program {
    public static int[] Puzzle(int[] a) {
        int[] b = a;
        for (int i = 1; i < b.Length / 2; i++) {
            int temp = b[i];
            b[i] = b[b.Length - 1 - i];
            b[b.Length - 1 - i] = temp;
        }
        return b;
    }
}

This should be a zero

This should be a <

This should be -- instead of ++

public class Program {
    public static int[] Puzzle(int[] b) {
        int front, back, temp;
        int[] a = b;
        front = 0;
        back = a.Length - 1;
        temp = a[back];

        while (front > back) {
            a[back] = a[front];
            a[front] = temp;
            ++back;
            ++front;
            temp = a[back];
        }

        return a;
    }
}
Computer Augmented Program Engineering

- 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!