Accelerating Legacy String Kernels via Bounded Automata Learning

Kevin Angstadt†, Jean-Baptiste Jeannin‡, Westley Weimer†

†Computer Science and Engineering, University of Michigan
‡Aerospace Engineering, University of Michigan

This work is funded in part by: NSF grants CCF-1629450, CCF-1763674, CCF-1908633; AFRL Contract No. FA8750-19-1-0501; and the Jefferson Scholars Foundation.
Physical Limits Spark Creativity

Hardware accelerators are seen as a viable path forward for tackling increasing compute demands.

Source: MIT Technology Review

New Kinds of Processors
New Kinds of Processors

Xilinx provides Alibaba Cloud FaaS with AI Acceleration

Why Microsoft Has Bet on FPGAs to Infuse Its Cloud With AI

EC2 F1 Instances with FPGAs – Now Generally Available
Legacy Code in the Age of Hardware Accelerators

- Legacy code typically cannot be directly compiled for FPGAs
- Learning a new programming model is costly and slows rate of adoption of new accelerators
- May want to “try out” new hardware with existing software
  - No training on new hardware
  - Limited time or resources to allocate
Talk Overview

• Background and Motivation
• Technical Approach
  • High-Level Summary
  • Problem Statement
  • Approach Details
  • Formal Results
• Empirical Evaluation
• Open Challenges

Goal: Aid developers programming FPGAs by automatically porting certain classes of existing source code without requiring low-level hardware knowledge to produce performant code
AutomataSynth at a Glance

• Framework for executing code (legacy software) on FPGAs and other hardware accelerators
• Dynamically observe and statically analyze program behavior to synthesize a functionally-equivalent hardware design
• Initial effort infers a set of finite automata rather than attempting to directly compile code
• Novel combination of model learning (learning theory), software model checking (software engineering), string decision procedures (PL/theory), and high-performance automata architectures (hardware)
Why Automata(Synth)?

• FPGA designs are often described in terms of state machines
• Automata a versatile and broadly-applicable
• Can build on significant research effort for accelerating state machine execution
• Other high-level approaches (cf. HLS) generally fail to abstract low-level architectural details
• Our approach decouples high-level program and low-level implementation
Automata Accelerate Big Data Applications

- Detecting Intrusion Attempts in Network Packets
- Learning Association Rules with an *a priori* approach
- Detecting incorrect POS tags in NLP
- Looking for Virus Signatures in Binary Data
- Detecting Higgs Events in Particle Collider Data
-Aligning DNA Fragments to the Human Genome
Problem Statement (First Efforts)

• Input: function \texttt{kernel : string \rightarrow bool}

• Assumptions:
  • Function decides a \texttt{regular language}
  • Source code for function is available

• Output: finite automaton with the same behavior on “all” inputs as kernel
Angluin-Style Learning (L*)

Learner

Teacher

Oracle

Membership Query

$s \in L(\text{Kernel})$

Termination Query

$L(M) \overset{?}{=} L(\text{Kernel})$

Automaton $M$

Yes/No Answer

Yes or Counterexample
Membership Queries are Direct

\[ s \in L(Kernel) \]

- Check if kernel accepts input by **running the code**
- Return value of the kernel is the answer from the teacher
- **Caution:** take care with ASCII encoding and null terminators (not all functions assume C-style strings)
Understanding Termination Queries

$L(M) = L(Kernel)$

- Don’t have held-out automaton for comparison
- Test inputs generally do not suffice
  - Coverage, generation, etc. difficult challenges
- Constraint over string inputs
  - No inputs that are accepted by the kernel are rejected by the candidate machine (and vice versa)
  - “The symmetric difference is empty”
  - Allows for formulation as a software verification query
Equality Checking as Software Verification

• Explores control flow graph looking for property violations
  • Success finding variety of bugs (e.g., double-free, locking violations, etc.)
  • Used in industry for driver verification

• Bounded Model Checking suitable for this domain
  • Verifies that property holds for all program executions up to length k (i.e., fixed number of loop unrollings)
  • Incremental unrolling to check longer and longer executions
  • Use theorem prover to identify executions that violate property

• Wrapper program to encode the “symmetric difference” property

• Add in string solver to generate counterexamples
AutomataSynth System Architecture

- **L* Learner**
  - \( s \in L(Kernel) \) (Membership Query)
  - \( L(M) \approx L(Kernel) \) (Termination Query)

- **Mapper**

- **Kernel**

- **Synthesis**

- **Software Verifier**
  - **SMT Solver**
  - **String Solver**

- **Learned Automaton** \( \mathcal{M} \)

- **FPGA**
Caveats/Challenges

Theorem provers are relatively complete.

• Software verification will occasionally return an unknown result
• No counterexample is produced, so L* cannot continue
• Implication: resulting automaton is approximate, but correct for all inputs shorter than some fixed bound

BMC with incremental unrolling is a semi-algorithm.

• Unrolling of program with infinite loops could continue indefinitely
• Termination query might never terminate
• For regular languages finite unrolling suffices (See §4.3)
• Implication: BMC+string solver will terminate and satisfies requirements for Termination Queries
When AutomataSynth terminates, we report if the automaton is **correct** or **approximate**

- Formal approach means that **automata are provably correct**
- Approximate automata are correct for inputs up to a **known bound**

For functions deciding a regular language, **correctness is guaranteed** (modulo the theorem prover)

In practice, we make use of **timeouts** to terminate AutomataSynth

- Tunable to help define bounds of correctness
Evaluation: Guiding Research Questions

• How many real-world string kernels can AutomataSynth correctly learn? With approximation?

• Does AutomataSynth learn automata that fit within the design constraints of modern, automata-derived, reconfigurable architectures?
Experimental Methodology

- Mine GitHub for string functions in top C repositories
- Use Cil framework to iteratively parse each source file and extract all string functions
- Filter for duplicates and manual analysis to filter on Boolean return type
- Considered 26 repositories, 973 separate string functions, 18 meaningfully-distinct real-world benchmarks
  - AutomataSynth did not support 3 due to functionality of underlying string solver (e.g., no math on characters)
## Empirical Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Project</th>
<th>LOC</th>
<th>Member Queries</th>
<th>Term. Queries</th>
<th>States</th>
<th>Runtime (min)</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>git_offset_1st_component</td>
<td>Git: Revision control</td>
<td>6</td>
<td>4,090</td>
<td>2</td>
<td>2</td>
<td>0.12</td>
<td>✓</td>
</tr>
<tr>
<td>checkerrormsg</td>
<td>jq: Command-line JSON processor</td>
<td>4</td>
<td>32,664</td>
<td>2</td>
<td>15</td>
<td>1436.58</td>
<td>✓*</td>
</tr>
<tr>
<td>checkfail</td>
<td></td>
<td>14</td>
<td>189,013</td>
<td>3</td>
<td>35</td>
<td>1438.47</td>
<td>✓*</td>
</tr>
<tr>
<td>skipline</td>
<td></td>
<td>17</td>
<td>7,663</td>
<td>3</td>
<td>3</td>
<td>4.90</td>
<td>✓</td>
</tr>
<tr>
<td>end_line</td>
<td>Linux: OS kernel</td>
<td>11</td>
<td>510,623</td>
<td>4</td>
<td>44</td>
<td>491.88</td>
<td>✓</td>
</tr>
<tr>
<td>start_line</td>
<td></td>
<td>11</td>
<td>206,613</td>
<td>2</td>
<td>46</td>
<td>80.22</td>
<td>Approx.</td>
</tr>
<tr>
<td>is_mcounted_section_name</td>
<td>MASSCAN: IP port scanner</td>
<td>54</td>
<td>672,041</td>
<td>7</td>
<td>57</td>
<td>1439.98</td>
<td>Approx.</td>
</tr>
<tr>
<td>is_numeric_index</td>
<td></td>
<td>17</td>
<td>10,727</td>
<td>3</td>
<td>4</td>
<td>4.95</td>
<td>✓</td>
</tr>
<tr>
<td>is_comment</td>
<td></td>
<td>11</td>
<td>4,090</td>
<td>2</td>
<td>2</td>
<td>0.23</td>
<td>✓</td>
</tr>
<tr>
<td>AMF_DecodeBoolean</td>
<td>OBS Studio: Live streaming and recording software</td>
<td>2</td>
<td>2,557</td>
<td>2</td>
<td>2</td>
<td>0.07</td>
<td>✓</td>
</tr>
<tr>
<td>cf_is_comment</td>
<td></td>
<td>28</td>
<td>4,599</td>
<td>2</td>
<td>4</td>
<td>5.00</td>
<td>✓</td>
</tr>
<tr>
<td>cf_is_splice</td>
<td></td>
<td>22</td>
<td>1,913</td>
<td>2</td>
<td>4</td>
<td>0.05</td>
<td>✓</td>
</tr>
<tr>
<td>is_reserved_name</td>
<td></td>
<td>39</td>
<td>240,705</td>
<td>8</td>
<td>42</td>
<td>1424.48</td>
<td>✓</td>
</tr>
<tr>
<td>has_start_code</td>
<td></td>
<td>18</td>
<td>10,213</td>
<td>2</td>
<td>7</td>
<td>0.08</td>
<td>✓</td>
</tr>
<tr>
<td>stbtt__isfont</td>
<td>Openpilot: Open-source driving agent</td>
<td>24</td>
<td>79,598</td>
<td>5</td>
<td>19</td>
<td>0.22</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Empirical Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Project</th>
<th>LOC</th>
<th>Member</th>
<th>Term.</th>
<th>Queries</th>
<th>Runtime (min)</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>git_offset_1st_component</td>
<td>Git: Revision control</td>
<td>6</td>
<td>4,090</td>
<td>2</td>
<td>2</td>
<td>0.12</td>
<td>✔</td>
</tr>
<tr>
<td>checkererrmsg</td>
<td>jq: Command-line JSON processor</td>
<td>4</td>
<td>32,664</td>
<td>2</td>
<td>15</td>
<td>1436.58</td>
<td>✔*</td>
</tr>
<tr>
<td>checkfail</td>
<td></td>
<td>14</td>
<td>189,013</td>
<td>3</td>
<td>35</td>
<td>1438.47</td>
<td>✔*</td>
</tr>
<tr>
<td>skipline</td>
<td></td>
<td>17</td>
<td>7,663</td>
<td>3</td>
<td>3</td>
<td>4.90</td>
<td>✔</td>
</tr>
<tr>
<td>end_line</td>
<td>Linux: OS kernel</td>
<td>11</td>
<td>510,623</td>
<td>4</td>
<td>44</td>
<td>491.88</td>
<td>✔</td>
</tr>
<tr>
<td>start_line</td>
<td></td>
<td>11</td>
<td>206,613</td>
<td>2</td>
<td>46</td>
<td>80.22</td>
<td>Approx.</td>
</tr>
<tr>
<td>is_mcounted_section_name</td>
<td></td>
<td>54</td>
<td>672,041</td>
<td>7</td>
<td>57</td>
<td>1439.98</td>
<td>Approx.</td>
</tr>
<tr>
<td>is_numeric_index</td>
<td>MASSCAN: IP port scanner</td>
<td>17</td>
<td>10,727</td>
<td>3</td>
<td>4</td>
<td>4.95</td>
<td>✔</td>
</tr>
<tr>
<td>is_comment</td>
<td></td>
<td>11</td>
<td>4,090</td>
<td>2</td>
<td>2</td>
<td>0.23</td>
<td>✔</td>
</tr>
<tr>
<td>AMF_DecodeBoolean</td>
<td></td>
<td>2</td>
<td>2,557</td>
<td>2</td>
<td>2</td>
<td>0.07</td>
<td>✔</td>
</tr>
<tr>
<td>cf_is_comment</td>
<td></td>
<td>28</td>
<td>4,599</td>
<td>2</td>
<td>4</td>
<td>5.00</td>
<td>✔</td>
</tr>
<tr>
<td>cf_is_splice</td>
<td></td>
<td>22</td>
<td>1,913</td>
<td>2</td>
<td>4</td>
<td>0.05</td>
<td>✔</td>
</tr>
<tr>
<td>is_reserved_name</td>
<td></td>
<td>39</td>
<td>240,705</td>
<td>8</td>
<td>42</td>
<td>1424.48</td>
<td>✔</td>
</tr>
<tr>
<td>has_start_code</td>
<td></td>
<td>18</td>
<td>10,213</td>
<td>2</td>
<td>7</td>
<td>0.08</td>
<td>✔</td>
</tr>
<tr>
<td>stbtt__isfont</td>
<td>Openpilot: Open-source driving agent</td>
<td>24</td>
<td>79,598</td>
<td>5</td>
<td>19</td>
<td>0.22</td>
<td>✔</td>
</tr>
</tbody>
</table>

AutomataSynth learns 13/18 kernels correctly and a further 2 approximately.
Learning took an average of 7 hours. More than half take fewer than 5 minutes.
## Empirical Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Project</th>
<th>LOC</th>
<th>Member Queries</th>
<th>Term. Queries</th>
<th>States</th>
<th>Runtime (min)</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>git_offset_1st_component</td>
<td>Git: Revision control</td>
<td>6</td>
<td>4,090</td>
<td>2</td>
<td>2</td>
<td>0.12</td>
<td>✔</td>
</tr>
<tr>
<td>checkererrormsg</td>
<td>jq: Command-line JSON processor</td>
<td>4</td>
<td>32,664</td>
<td>2</td>
<td>15</td>
<td>1436.58</td>
<td>✔️</td>
</tr>
<tr>
<td>checkfail</td>
<td></td>
<td>14</td>
<td>189,013</td>
<td>2</td>
<td>35</td>
<td>1438.47</td>
<td>✔️</td>
</tr>
<tr>
<td>skyline</td>
<td></td>
<td>17</td>
<td>7,663</td>
<td>3</td>
<td>3</td>
<td>4.90</td>
<td>✔</td>
</tr>
<tr>
<td>end_line</td>
<td></td>
<td>11</td>
<td>510,623</td>
<td>4</td>
<td>44</td>
<td>491.88</td>
<td>✔</td>
</tr>
<tr>
<td>start_line</td>
<td>Linux: OS kernel</td>
<td>11</td>
<td>206,613</td>
<td>2</td>
<td>46</td>
<td>80.22</td>
<td>Approx.</td>
</tr>
<tr>
<td>is_mcounted_section_name</td>
<td></td>
<td>54</td>
<td>672,041</td>
<td>7</td>
<td>57</td>
<td>1439.98</td>
<td>Approx.</td>
</tr>
<tr>
<td>is_numeric_index</td>
<td>MASSCAN: IP port</td>
<td>17</td>
<td>10,727</td>
<td>3</td>
<td>4</td>
<td>4.95</td>
<td>✔</td>
</tr>
<tr>
<td>is_comment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>0.23</td>
<td>✔</td>
</tr>
<tr>
<td>AMF_DecodeBoolean</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>0.07</td>
<td>✔</td>
</tr>
<tr>
<td>cf_is_comment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>5.00</td>
<td>✔</td>
</tr>
<tr>
<td>cf_is_splice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>0.05</td>
<td>✔</td>
</tr>
<tr>
<td>is_reserved_name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>42</td>
<td>1424.48</td>
<td>✔</td>
</tr>
<tr>
<td>has_start_code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>0.08</td>
<td>✔</td>
</tr>
<tr>
<td>stbtt__isfont</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>19</td>
<td>0.22</td>
<td>✔</td>
</tr>
</tbody>
</table>

Learned automata fall within resource constraints of FPGA-based architectures.
Open Challenges: Guiding Desires

• Support broader classes of functions
  • Not all legacy code consists solely of Boolean string kernels
• More indicative benchmark applications
  • Mined real-world functions are a start, but...
  • Functions that dominate runtime
  • Larger or more complex functions
• Comparison of learning-based solutions (AutomataSynth) with HLS
Open Challenges (Learning-Specific)

1. Learn More Expressive Models (Transducers, Pushdown, etc.)
2. Improve String Solvers (Expressiveness, Performance, etc.)
3. Scale Termination Queries and Explore Alternatives
4. Understand Approximation (PAC Learning, Measure Error)
AutomataSynth Summary

• Framework for accelerating legacy Boolean string kernel functions using FPGAs

• Static and dynamic analyses of program behavior to construct functionally-equivalent automata

• Novel combination of Angluin-style learning with software model checking and string solvers

• Successfully constructs equivalent (or near equivalent) FPGA designs for more than 80% of real-world benchmarks mined from GitHub

• Many open challenges mean many opportunities for studying learning-based approaches for porting code

Source code: https://github.com/kevinaangstadt/automata-synth