Special Circumstances, LLC

In response to: Recovery of Symbolic Mathematics from Code (ReMath)

Proposal title: Reverse Engineering Boolean Functions Using constrained Symbolic Regression (REFUSR)
September 17, 2020

Lead proposer organizational name: Special Circumstances, LLC
Type of business: WOMEN-OWNED SMALL BUSINESS
Other Team Members: NONE
Technical point of contact:

Lucca Fraser
Special Circumstances
P.O. Box 221812
Anchorage, AK 99522

Email: lucca.fraser@special-circumstanc.es
Phone: +1 (902) 222-7378
Fax: NONE

Administrative point of contact:

Meredith L. Patterson
Special Circumstances
P.O. Box 221812
Anchorage, AK 99522

Email: mlp@special-circumstanc.es
Phone:
Fax: NONE

Table of Contents

I. EXECUTIVE SUMMARY

ReMath asks if it is possible to develop methodologies or technologies to analyze mathematical functions in programs used to control cyber-physical systems. Rather than approaching this as a problem for *automatic translation*, we have come to view it instead as a problem of *analytically constrained symbolic regression*. The path forward, we believe, lies in a combination of evolutionary program synthesis with constraint-based programming. Among the strengths of this approach is that it does *not* require us to first assemble a massive dataset of examples of mathematical specifications and their cyber-physical implementations. Instead, *for each* cyber-physical implementation under examination, we will generate (1) a dataset comprising a large number of *input/output pairs*, generated by feeding random parameters into the implementations and monitoring the output (or behaviour), and (2) a knowledge base of *constraints* pertaining to the implementation, obtained through various analytical means (including, but not necessarily limited to: techniques from the analysis of boolean functions (ABF), static binary and source code analysis, and timing information).

To recover symbolic representations of mathematical functions from their cyber-physical implementations, we thus begin by treating the CPS as a black box, mapping inputs to outputs. By feeding a series of random inputs into the CPS, we obtain a partial graph. At this point, our problem already has a familiar form to it: finding a function whose graph fits the available data points is a matter of *symbolic regression*.[1] But here we may take advantage of the fact that the CPS is not, in fact, a black box. Through existing static analysis techniques, including property-testing methods originating in ABF, we may infer a number of formal constraints. This provides an opportunity to draw on the recent work of Iwo Błądek and Krzysztof Krawiec on what they have called *counterexample-driven genetic programming* (CDGP) and the related technique of *symbolic regression with formal constraints* (SRFC).[2]

II. TECHNICAL DESCRIPTION

Our intention is to begin with the relatively simple case of reverse engineering a set of Boolean functions, implemented as programs for the ACE 11 programmable logic controller (PLC) (discussed in some detail, below). Faced with the task of discovering which Boolean function,

---

[1] John R. Koza; Martin A. Keane; James P. Rice (1993). "Performance improvement of machine learning via automatic discovery of facilitating functions as applied to a problem of symbolic system identification". IEEE International Conference on Neural Networks. San Francisco: IEEE. pp. 191–198. See also Patryk Orzechowski, William La Cava, and Jason H. Moore, "Where are we now? A large benchmark study of recent symbolic regression methods", arXiv:1804.09331v2 [cs.NE], presented at GECCO '18, for an idea of what sort of performance we might plausibly expect from such methods.

[2] Iwo Błądek and Krzysztof Krawiec, "Solving symbolic regression problems with formal constraints", GECCO '19: Proceedings of the Genetic and Evolutionary Computation ConferenceJuly 2019 Pages 977–984https://doi.org/10.1145/3321707.3321743

for example, is being implemented by a fragment of PLC code F, we shall proceed in the following fashion:

1. We begin by creating an initial set of tests **T** for this population. These are produced by generating a random set of inputs for **F** and collecting the outputs.
2. We then, through various analytic means, establish a set of formal properties that **F** is either certain or extremely likely to possess. These are then expressed as a set of constraints **C** that any candidate must satisfy if it is to be counted as equivalent to **F**.
3. We then generate an initial population **P** of randomly-generated symbolic expressions, each of which evaluate to a Boolean function.
4. Next, we perform a form of *lexicase selection*[3] on **P**: iterating through **P** in random order, we set aside any that either fail to pass some subset of tests in **T**, gradually increasing the size of this subset, until only n such candidates c remain.
5. We then enlist a SMT solver (Z3, for example) to try to generate an input that, when passed to some such candidate, violates one or more of the constraints in **C**. This input -- a counterexample to our candidate solutions -- is then fed to **F**, yielding a new test, which is appended to **T**.
6. If, for some candidate **p**, no such counterexample can be found, *and* all of the tests in **T** have been passed, then we are finished: **p** is then taken as a symbolic expression of the function implemented in **F**. Go to step 9.
7. If not, we apply one or more genetic operators, such as crossover or mutation, to the remaining candidates, and insert the resulting n offspring into **P**, replacing whichever n individuals were first eliminated by the lexicase selection process.
8. We then go back to step 4, and repeat the cycle until a perfect solution to every test pair in **T**, and every constraint in **C**, is found.
9. Once a solution is found, we test it against a battery of fresh input/output pairs from **F**, to gauge the accuracy of our search. If inaccuracies are detected, add the tests it failed to **T**, and go to step 4.

The programs in **P** will be represented as purely functional symbolic expressions. These are still a few steps away from providing useful mathematical summaries to the subject matter expert (SME), owing to the peculiarly "hairy" and irregular character of evolved programs. Fortunately, the automatic simplification of symbolic expressions is a well-researched domain.[4] The simplified symbolic expressions we arrive at should be essentially human-readable, and meaningfully interpretable by a human SME. A concise statement of the formal constraints that

---

[3] Thomas Helmuth, Lee Spector, and James Matheson. 2015. "Solving Uncompromising Problems with Lexicase Selection". IEEE Transactions on Evolutionary Computation 19, 5 (Oct. 2015), 630–643. https://doi.org/doi:10.1109/TEVC.2014.2362729. Our choice of lexicase selection is motivated by two considerations: (1) we are dealing, here, with an exact or "uncompromising" problem, where approximate solutions are of little interest, other than gesturing towards the desired solution, and (2) Helmuth, Spector, and Matheson have convincingly shown lexicase to offer some resistance to overfitting, and to better facilitate the discovery of simple solutions.

[4] David H. Bailey, Jonathan M. Borwein, Alexander D. Kaiser, "Automated simplification of large symbolic expressions", *Journal of Symbolic Computation*, Volume 60, January 2014, pp 120-136.

these expressions satisfy will be included in the system's reports as well, to provide further context, and to allow the SME to better understand the system's reasoning.

The strength of this approach lies in its flexibility. We are free to use any means at our disposal to infer properties of the CPS program in question, so long as these properties can be translated into the language of formal constraints, expressed as SMT formulae.

One such method, which we are currently exploring, draws on the analysis of Boolean functions (ADF), which supplies techniques for locally testing arbitrary Boolean functions for general formal properties. The Blum-Luby-Rubinfeld test, for instance, provides a means for determining whether an arbitrary Boolean function f is either linear, or approximately linear, by choosing two inputs x and y at random, and testing whether f(x) + f(y) = f(x + y). Through the repeated selection of inputs, we may further infer how near f is to being linear, since this proximity will be proportional to the probability of f(x) + f(y) being equal to f(x+y) for arbitrary x and y.[5]

Bounds on the complexity of candidate functions, moreover, could plausibly be inferred either through the static analysis of **F**'s source or binary code, or even by gathering timing information as **F** processes various inputs.

The idea, here, is ultimately fairly simple: rather than searching for a "silver bullet" that will let us solve the recovery problem in a single stroke, we begin with the modest but firmly established technique of recovering unknown functions through symbolic regression, while sharpening the focus of our search with successive layers of formal constraints, obtained through an array of heterogeneous techniques. Błądek and Krawiec's CDGP technique allows us to bring this motley arsenal into the unity of a coherent evolutionary search.

III. OUR CYBER-PHYSICAL TARGET: THE ACE 11 PROGRAMMABLE LOGIC CONTROLLER

Programmable Logic Controllers (PLCs) are digital devices used to control cyber-physical systems such as pumps, valves, conveyor belts, manipulators, and environmental controls. A program written in a low-level language is used to change the state on the I/O pins of the PLC, which in turn actuate some sort of cyber-physical device. PLC design is typically heavy on control lines with low memory and modest computational power. Their widespread use in long-lived or legacy industrial systems are, in essence, the “cyber” in “cyber-physical systems” .

The ACE 11 PLC is a low-cost Common Off-The-Shelf (COTS) device capable of controlling a variety of industrial equipment that are considered cyber-physical targets. It is a 6 digital input and 6 digital output PLC, that runs on a Texas Instrument Thumb-2 ARM microcontroller SoC (TM4C1231H6PM), and implements a Cortex-M4F core which is designed for small memory footprint embedded devices. The device is programmed using a system called vBuilder, developed by its manufacturer, Velocio. The programming language can be expressed either through traditional ladder logic, or a "click-n-drag" graphical flowchart style, similar to the

[5] For details, see Ryan O'Donnell, The Analysis of Boolean Functions, Cambridge: Cambridge University Press, 2014.

application MATLAB. Code is compiled on a local host system and then uploaded to the ACE11 over its USB interface.

While the ACE 11 is physically smaller than many PLCs “in the wild”, it has all the features and functionality of a full-fledged PLC. This means that techniques developed to reverse engineer mathematical functions implemented on the ACE 11 should, in principle, generalize to larger, more complex PLCs, that use the same type of programming.

The hardware design of the ACE 11 is more or less standard. The general design of most PLCs is around some type of embedded microcontroller that has some number of digital and/or analogue inputs and outputs, which themselves can be used to physically connect the PLC to any number of industrial systems. It supports modbus, a common control protocol used in SCADA systems to facilitate communication with sensors, motors and other components of various industrial processes.

VI. STATEMENT OF WORK (SOW)

Phase 1	Milestone 1
Objective: Plan Development: Develop detailed experimental plan for the identified SME use cases. The plan must identify domain use case(s) and representations to be used, outline AI approaches to explore, and their training sources, and detail approaches to automated augmentation and generation of training corpora.	Detailed Experiment Plan

Approach: We focus the REFUSR research program on studying the applicability of *counterexample-driven genetic programming* (CDPG) to efficient *symbolic regression with formal constraints* (SRFC) on ACE 11 PLC devices implementing boolean functions.	
Exit Criteria: Delivery of a Detailed Plan of Experiments	

Phase 1	Milestone 2
Objective: Data Set Generation: Identify reliable mathematical primitives aided by source-level representations. The report for this milestone will include refinement of automatic generation of training data.	Milestone Report which identifies primitives

Phase 1	Milestone 2
Approach: We develop and apply a high-certainty, low-efficiency variant of the CDPG-inspired symbolic regression procedure described in section II of this research proposal to generate a database of formulae from a database of ACE 11 devices implementing 'PLC-typical' source code with a boolean output. Note that our CPDG-inspired symbolic regression procedure is, by design, not 'big data' reliant. This bears on the nature of our approach to Milestone 2: The function of the selected ACE 11 database and the generated database of high-certainty extracted formulae is merely to provide us with (assumed) ground-truths to test the high-efficiency algorithm we subsequently develop to meet Milestone 3. A major benefit of the approach described above is that our database-generation workflow will function as a relaxed "first pass" at realizing our CPDG-inspired formulae extraction procedure, suspending concerns with efficiency and focussing on reliability. A different, simpler approach to database generation for REFUSR also bears mentioning: the simple act of implementing boolean functions of our choice in ACE 11 source code, which should be easy to automate for certain classes of boolean functions, can provide us with unlimited fully-certain ground truths. We may choose to produce both a 'source-code first' database and a 'function first' database, employing both approaches.	
Exit Criteria: Generated database of formulae	

Phase 1	Milestone 3
Objective: Demonstration: Extract formulaic expressions aided by source-level representations. The report for this milestone will include an initial demonstration of integration with modular program analysis tools, providing SME-aiding automation in the selected use case(s).	Milestone Report including Initial Demonstration
Approach: We develop a high-efficiency variant of the	

CDPG-inspired symbolic regression procedure described in section II of this research proposal and apply it to real-time formula extraction from our ACE 11 devices dataset. We treat the previously generated high-certainty formulae from Milestone 2 as ground truth. Note that modular program analysis plays a core role in our method, as the constraint-derivation stage of the procedure we describe in section II is essentially an open-ended automated program analysis stage. A successful demonstration of our CDPG-inspired method is therefore a fortiori a demonstration of integration with modular program analysis tools.	
Exit Criteria: Demonstration of Extraction of Formulaic Expressions	

Phase 1	Milestone 4
Objective: Case Studies: Extract SME domain communications aided by source-level representations. The report should include successful case studies with application of AI methods, methodological summaries, algorithms, datasets, and finalized formalisms consolidated in a report. For unsuccessful case studies, produce a report of the encountered obstacles and summaries of the approaches attempted. End of Phase I report is produced.	Phase I Report
Approach: We produce two technical reports. We pay special attention to the role of access to source code in the constraint-derivation stage of our formula extraction procedure in successful case studies.	
Exit Criteria: Phase I Report Delivered	

Phase 2	Milestone 5

Objective: Produce Automated Training Corpora: Identify mathematical primitives from binary-level code, unaided by source-level representations. Produce a report for this milestone that includes details of automated training corpora augmentation and generation for binary code.	Milestone Report including details of training corpora and generation
Approach: We iterate the high-certainty formulae generation procedure from Milestone 2, this time restricting the constraint-derivation subprocedure to binary analysis. We compare results with our previously generated (via source code) assumed ground truths and adjust our process to compensate for the lack of source access, adding new binary analysis techniques as needed. Once our binary-only high-certainty variant of our procedure proves adequate, we move to iterate Milestone 3, developing a binary-only high-efficiency variant.	
Exit Criteria: Delivery of Milestone Report including details of training corpora and generation.	

Phase 2	Milestone 6
Objective: Demonstrated Integration: Extract formulaic expressions from binary-level code, unaided by source-level representations. Produce demonstration to demonstrate integration with modular binary analysis tools with a detailed case study of SME-aiding automation on binary code in the selected use cases.	Milestone Report including Demonstration
Approach: We note that successful application of our CDGP-inspired procedure to binary-level code is a fortiori successful integration with modular binary analysis tools, and focus on the construction of a realistic, practically relevant case study. We are considering various legacy industrial systems that run on older PLCs comparable in functionality to ACE 11 as possible candidates.	
Exit Criteria: Milestone Report delivered, Integrated Prototype is Demonstrated.	

Phase 2	Milestone 7
Objective: Final Prototype: Extract SME domain communications from binary-level representations, produce final prototype and report. Report should include successful case studies with application of AI methods, with methodological summaries, algorithms, datasets, and finalized formalisms consolidated in a report. For unsuccessful case studies, a report of the encountered obstacles and summaries of the approaches attempted.	Phase II Final Report and Prototype
Approach: As noted, the ACE 11 executes binary code on an ARM Thumb SOC, and so it should be possible to design an emulator that can evaluate binary code fragments with sufficient accuracy, by modifying the Thumb emulation module of the Unicorn CPU emulator library (a library to which our team has made extensive contributions in the past). This will eventually let us perform analyses without depending on a physical ACE 11 device. The symbolic expressions generated by our algorithm will be simplified and pretty-printed in a human readable fashion, along with the formal constraints discovered through the analytic process and used to guide the evolutionary search. These will be used to generate a report, for the eyes of a human SME.	
Exit Criteria: End of Phase II report is produced.	

VII. SCHEDULE, MILESTONES, AND DELIVERABLES

Special Circumstances has organized the project internally to include significant buffer time between tasks; this is to provide room should any complications arise (such as damaged equipment) and to provide staff with time to attend conferences and take holiday/vacation time off.

Summary of Important Dates:
We propose to start work on November 17, 2020. With that start date, the following table shows the projected dates of the milestones.

Event	Date	Cost
Project Start	November 17 2020	

Milestone 1	December 17 2020	$124,193.48
Milestone 2	February 17 2021	$100,764.63
Milestone 3	May 2021	$149,689.75
Milestone 4	August 2021	$124,358.57
Milestone 6	November 2021	$169,246.94
Milestone 7	February 2022	$180,416.17
Milestone 8	May 2022	$150,042.23
Project End	May 2022	

VIII. COSTS

This project expenses are split between labor and the costs associated with the exploration of our reference objects. The total expected project cost is approximately $980,00.00, including relevant taxes.

Labor Costs:
Meredith Patterson and Lucca Fraser have established rates of $75.00/hr, in both private sector work as well as past and current DARPA engagements. This project will also need an additional 3 personnel whose established rates are $62.50/hr, in both private sector work as well as current DARPA engagements. Between these team members, the project proposes 18 months of effort.
Meredith Patterson & Lucca Fraser: $75.00/hr x 388 days x 1,933 hours = $144,975.00
3 additional personnel: $62.50/hr x 388 days x 7,602 hours = $570,150.00

The total cost of labor is $715,125.00

Travel:
Special Circumstances will be conducting three total trips as part of the requirements of the project. Two trips in Washington, D.C. and one San Diego, CA. Average costs of round trip plane tickets for the four necessary SC personnel to these locations, hotels and per diem (rates from GSA.gov were applied) came to:

Trip to Washington , D.C. (1): Flight + Hotel + Per Diem = $9,016.00
Trip to Washington, D.C. (2): Flight + Hotel + Per Diem = $9,016.00
Trip to San Diego: Flight + Hotel + Per Diem = $8,972.00

Sub-proposer: NONE

Team-member identification:

Lucca Fraser, Canadian Citizen, not affiliated with any FFRDC
Meredith Patterson, US Citizen, not affiliated with any FFRDC
Steve Okay, US Citizen, not affiliated with any FFRDC
Peli Grietzer, Israel & Poland Citizen, not affiliated with any FFRDC
Daniel Kuehn, Spain, not affiliated with any FFRDC

Government or FFRDC team members: NONE

Organizational Conflict of Interest Affirmation and Disclosure: NONE

Human Use: NONE